Título:	Estudio e implementación de un framework de desarrollo de aplicaciones con funciones de seguridad y privacidad para móviles.
Autor:	Adrià Navarro Martin
Director:	Francisco Jordán Fernández
Departamento:	Arquitectura de computadores
Centro:	Facultat d'informàtica de Barcelona (FIB)
Universidad:	Universitat Politècnica de Catalunya (UPC) BarcelonaTech
Titulación:	Ingeniería Informática (2003)
Modalidad:	B
Empresa:	Safelayer Secure Communications S.A.
Fecha:	11/04/2013

AGRADECIMIENTOS

Con este proyecto termina una etapa muy importante de mi vida y me gustaría agradecerlo a todas las personas que me han acompañado y apoyado haciéndolo posible.

En primer lugar a mis padres, Teresa y Jordi, por darme la oportunidad de estudiar una carrera y apoyarme en mis decisiones.

A todos mis compañeros y amigos de la universidad, por acompañarme durante éste camino y hacer de él una experiencia inolvidable.

A todos los trabajadores de Safelayer, en especial Helena Pujol, Francisco Jordán y Alberto Guirao, por brindarme la oportunidad de realizar con ellos este proyecto y ayudarme y orientarme en incontables ocasiones.

A todos los compañeros becarios, Víctor, Adrián, Jesús, Iván, Xavi y Alberto con los que he hecho nuevas amistades y han hecho del entorno de trabajo una experiencia muy agradable.

Índice

I - INTRODUCCIÓN

1 Contexto

Este proyecto surgió como iniciativa de la empresa *Safelayer Secure Communications, S.A.* en Marzo de 2012 y ha sido realizado en el marco del proyecto AVANZA Privacy-aware Accountability for a Trustworthy Future Internet (PATFI) parcialmente financiado por el Ministerio de Industria, Turismo y Comercio.

*Safelayer Secure Communications, S.A.*es la compañía destacada en el mercado de seguridad y confianza para las TIC. Ofrece soluciones para la interpretación y generación de la confianza en los sistemas de información en entornos SOA y XML, tecnología de autenticación, firma electrónica y cifrado de datos para documentos y servicios web basada en PKI.

2 Motivación

En la actualidad, las plataformas móviles, como *Tablets* o *Smartphones* se usan de manera equiparable a las plataformas fijas como los ordenadores de sobremesa o portátiles. Sin embargo a día de hoy no podemos usar en nuestro dispositivo móvil muchos de los servicios a los que estamos acostumbrados y esto genera la necesidad de crear soluciones adaptadas a estas nuevas plataformas.

Las soluciones móviles corporativas permiten una alta flexibilidad para realizar operaciones de todo tipo desde cualquier lugar. Sin embargo no existen demasiadas aplicaciones orientadas a la seguridad y a la infraestructura de clave pública (PKI) que ofrezcan de manera cómoda y sencilla servicios como la firma electrónica o el cifrado.

Las principales plataformas para dispositivos móviles a día de hoy, que son iOS y Android, disponen de métodos a muy bajo nivel para realizar las principales acciones criptográficas, sin embargo resultan complicados de aplicar si no se tiene un alto entendimiento en la materia, especialmente si se desea crear una solución **multiplataforma**.

Para desarrollar una aplicación móvil multiplataforma existen varios *frameworks* que se benefician de HTML5 para reutilizar la totalidad o gran parte del código de una plataforma a otra, de manera que el tiempo de desarrollo queda reducido considerablemente. Estos *frameworks*, sin

embargo, carecen de librerías criptográficas que permitan desarrollar una aplicación orientada a la seguridad y la firma electrónica de manera relativamente sencilla.

3 Objetivos

El objetivo principal del proyecto es el "ESTUDIO E IMPLEMENTACIÓN DE UN *FRAMEWORK* DE DESARROLLO DE APLICACIONES CON FUNCIONES DE SEGURIDAD Y PRIVACIDAD PARA MÓVILES". Para ello:

i) se analiza el estado del arte en cuanto a plataformas de desarrollo móvil,

ii) se estudia el uso de tecnologías de seguridad basadas en criptografía, y entre ellas especialmente la de PKI (Public Key Infrastructure), así como protocolos de seguridad y privacidad que resulten aptos para móviles,

iii) se escoge un *framework* de desarrollo y plataformas que permitan el máximo despliegue en los móviles actuales, y

iv) se realiza un demostrador que valide el uso de dicho *framework*.

Como resultado final del proyecto se espera proporcionar el *framework* de desarrollo para móviles basados en iOS y Android, y la creación de un demostrador de **servicio de firma electrónica móvil multiplataforma** mediante el uso del *framework* anterior. El servicio podrá firmar documentos PDF con la herramienta open source para gestionar PDFs **iText**.

4 Resumen de resultados

El *framework* de desarrollo escogido es **PhoneGap**. PhoneGap es un *framework* que permite la creación de aplicaciones multiplataforma para diversos sistemas operativos móviles tales como iOS, Android, Windows Phone 7 y Blackberry entre otros. Se basa en el uso de **HTML5** y la integración de un navegador web que interprete ese código, incrustado en una aplicación nativa. Para las llamadas a sistema usa *plugins* específicos de cada plataforma. La extensión de este *framework* consiste en crear **plugins dedicados a operaciones criptográficas de firma** para las plataformas iOS y Android.

Sin embargo, la firma se podrá realizar tanto en el mismo dispositivo como utilizando un producto de la empresa Safelayer denominado **TrustedX**.

TrustedX es unaplataforma que proporciona servicios para la gestión de certificados y firma electrónica, cifrado de datos, y todos los protocolos auxiliares para el despliegue de aplicaciones que precisen PKI, mediante servicios web.

TrustedX también se usará como servicio de autenticación y proveedor de identidad (IdP) para los usuarios finales quienes, desde su teléfono y mediante el protocolo **OAuth 2.0**, se autenticarán contra *TrustedX*. Una vez autenticados tendrán acceso a sus documentos y a los servicios de firma.

Dado que el uso del servicio de firma estará orientado a dispositivos móviles, se busca minimizar el flujo de datos del teléfono al exterior y viceversa. Por lo que los documentos se almacenarán en la nube utilizando servicios como **Google Drive** o **Dropbox** y se descargarán y subirán mediante un servidor intermedio. Este calculará el hash del documento y lo mandará al servicio que lo tenga que firmar, sea el dispositivo móvil o *TrustedX* por lo que nunca llegará a transferirse la totalidad del documento para su firma.

Cuando el hash del documento esté firmado será mandado de vuelta al servidor intermedio, quien compondrá el documento PDF firmado y lo almacenará en la nube como un documento nuevo.

El uso del protocolo OAuth introduce las máximas garantías de privacidad entre el usuario móvil firmante y la aplicación.

II - CONCEPTOS BÁSICOS

1 Introducción a la seguridad informática

Dado que este proyecto se ubica en el campo de la seguridad informática, en esta sección se darán las bases necesarias para su correcto entendimiento.

Se explicarán los puntos más importantes, empezando por la Infraestructura de Clave Pública o como se dice mediante el acrónimo inglés, PKI. Después se explicarán los certificados digitales y la firma electrónica, siendo este último, el punto que más conexión tiene con este proyecto.

1.1 Criptografía

La base de la seguridad informática se sustenta en la **criptografía**. Ésta es la ciencia que estudia los métodos para poder convertir un mensaje entendible por cualquiera, en un mensaje inteligible mediante el uso de una **clave**. De esta manera el mensaje puede ser enviado de una forma **segura** ya que nadie puede descifrarlo y obtener su contenido, excepto el destinatario del mensaje y poseedor de la clave de que permita el **descifrado.** Esta clave puede ser la misma que se ha usado para el cifrado o no, dependiendo de si usamos criptografía **simétrica** o criptografía **asimétrica**.

La garantía de que el mensaje no sea descifrado dependerá de la seguridad de la **clave** que se esté usando. Esto es, que permanezca oculta, segura y sea de confianza. De manera que comprometer la clave implica comprometer el mensaje. Es por ello que hay que proteger las claves de accesos indeseados, almacenándolas en repositorios seguros y protegidos de posibles atacantes.

Esta ciencia nace de la necesidad de compartir información de modo confidencial, que sólo incumba a las partes implicadas. La criptografía incluye también el criptoanálisis, éste se encarga de intentar descifrar un mensaje para obtener la información que esconde en ausencia de la clave. Cuando no se puede recuperar y se llega a la conclusión que no se podría excepto si se tuviera la clave, se considera que se está utilizando un método de cifrado seguro.

Los métodos de cifrado son algoritmos matemáticos que permiten convertir texto llano en criptogramas ilegibles siendo necesaria la clave para descifrarlos y convertirlos de nuevo en texto llano.

1.1.1 Criptografía simétrica

La **criptografía simétrica**, también llamada criptografía de clave secreta, es un método criptográfico que se basa en el uso de la misma clave tanto para cifrar

como para descifrar el mensaje. Esto implica que los implicados en la comunicación deben ponerse de acuerdo sobre qué clave usar. Una vez ambos son conocedores de la clave, que debe acordarse de una manera segura y secreta (por eso la denominación *criptografía de clave secreta*), el remitente cifra el mensaje utilizando la clave, lo manda al destinatario que lo descifra utilizando la misma clave.

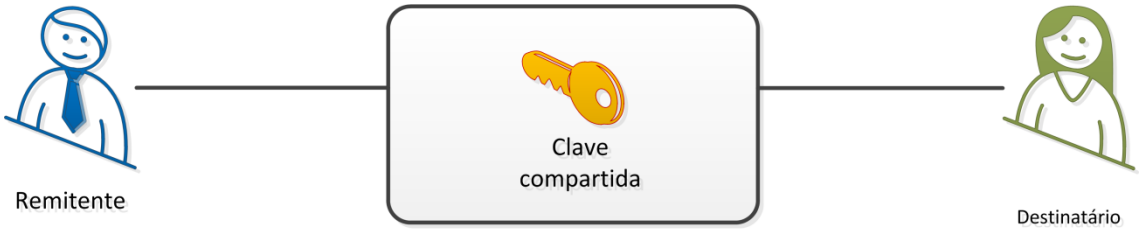


Figura II.1: Esquema de compartición de las claves

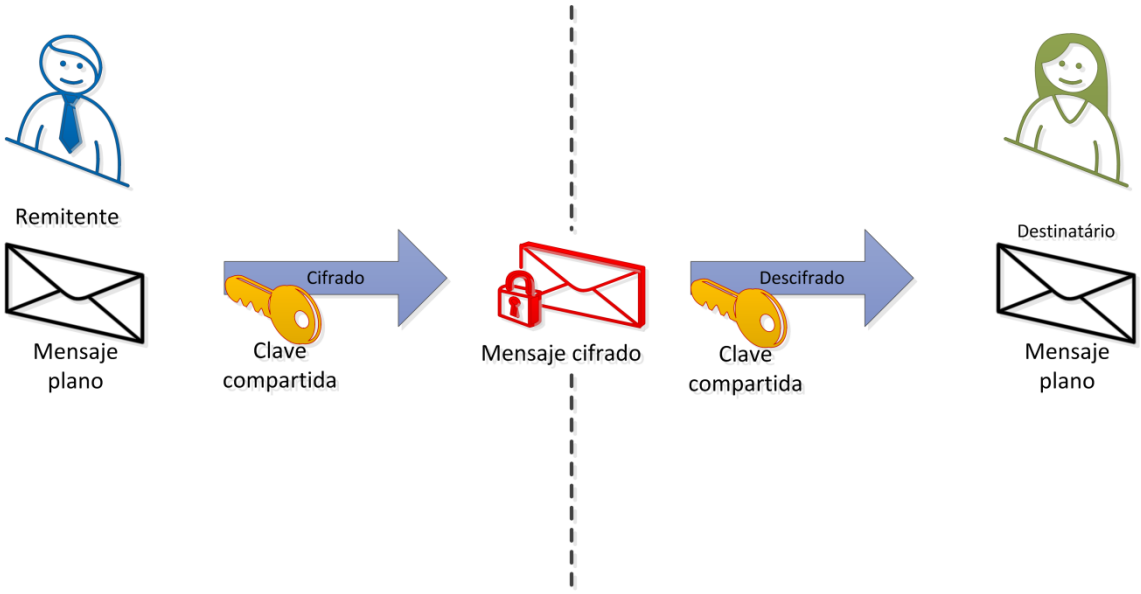


Figura II.2: Flujo de cifrado y descifrado mediante clave compartida

El proceso de cifrado tiene que garantizar que el mensaje se transforma de manera que no se pueda realizar el proceso inverso sin conocer la clave. La seguridad del método tiene que radicar en la clave en sí, y no en el algoritmo usado para ello. Es decir, conocer el algoritmo de cifrado no tiene que dar ninguna facilidad al criptoanálisis.

La criptografía simétrica garantiza la confidencialidad de la información que se transmite, sin embargo, presenta varios problemas:

- **Compartición de la clave:** Debido a que ambos implicados deben conocer la clave, el canal de comunicación para compartirla debe ser seguro ya que cualquier situación que comprometa la clave

comprometerá los futuros mensajes cifrados. Es por ello que normalmente debe haber un intercambio físico de la clave.

- **Gran cantidad de claves:** Cada par de individuos implicados en el envío de mensajes cifrados deberán disponer de una clave, por lo tanto hay una gran cantidad de claves a mantener de una forma segura.
- **Falta de autenticación:** Ambas partes utilizan la misma clave para la emisión de mensajes cifrados, esto implica que no se reconocerá con certeza el remitente del mensaje.

1.1.2 Criptografía asimétrica

La **criptografía asimétrica**, también llamada **criptografía de clave pública**, es un sistema de cifrado que se basa en el uso de un par por entidad para el proceso de cifrado y descifrado. En este par de claves hay una clave que es **pública**, es decir, que todo el mundo puede tener acceso a ella, y otra que es **privada**, a la que únicamente puede tener acceso la entidad poseedora de la clave.

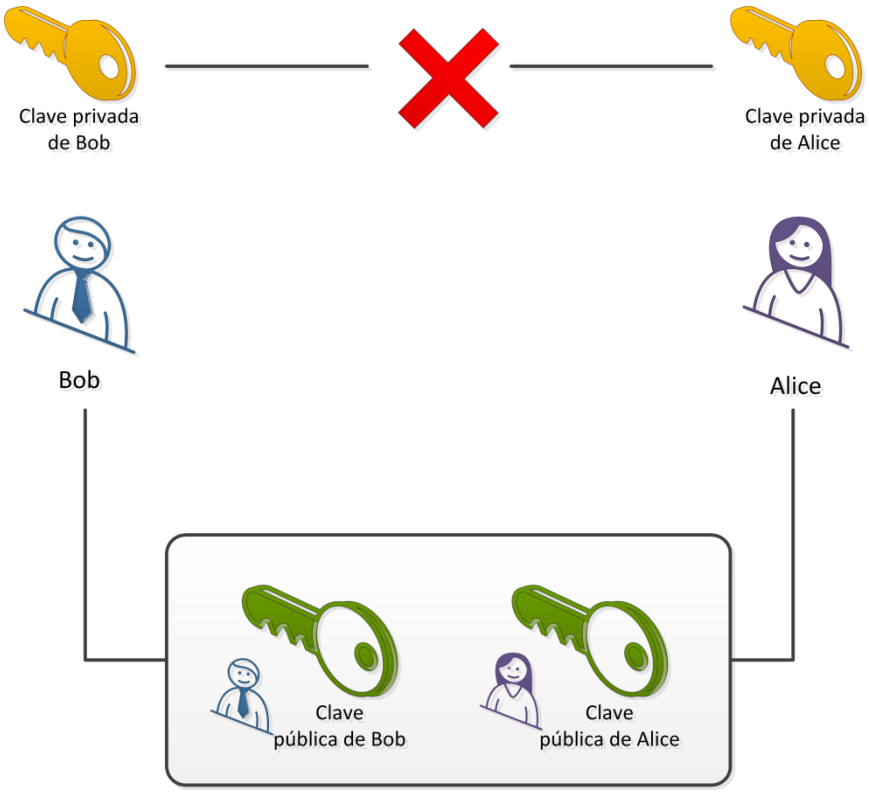


Figura II.3: Visibilidad de las claves en la criptografía asimétrica

Para el proceso de envío de mensajes cifrados, el remitente tiene que usar la clave pública del destinatario para cifrar el mensaje, una vez hecho esto, nadie más puede descifrarlo, excepto el destinatario, que usará su clave privada a la

que solamente él tiene acceso para volver a transformar el mensaje en texto plano.

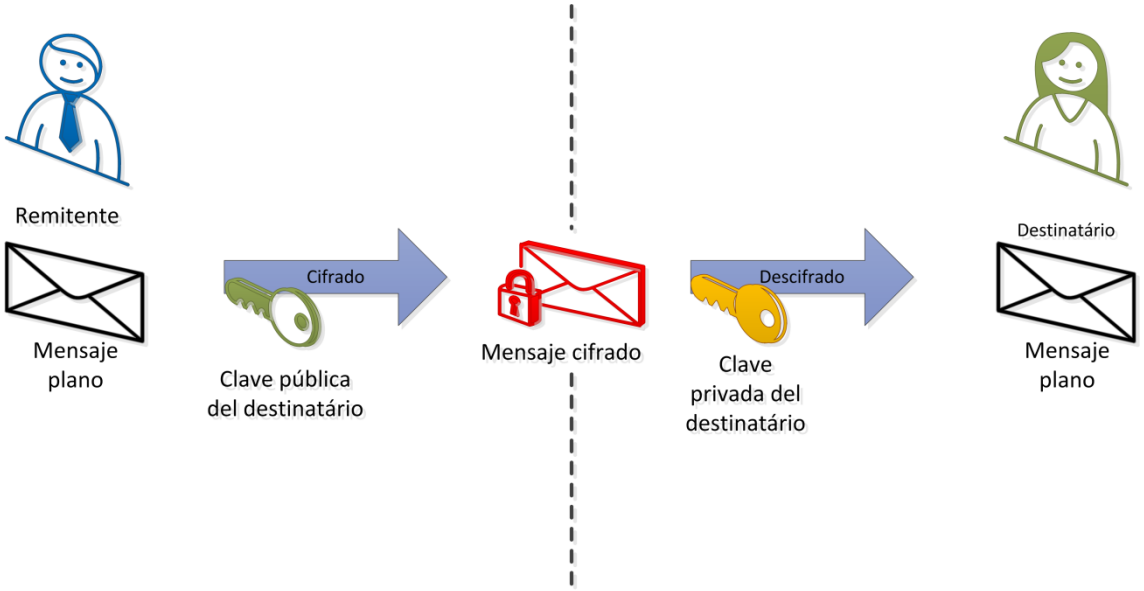


Figura II.4: Flujo de cifrado y descifrado mediante el par de claves

Ambas claves están matemáticamente relacionadas mediante funciones de un solo sentido que aprovechan propiedades particulares, por ejemplo de los números primos, por lo tanto es computacionalmente imposible hallar la clave privada a partir de la pública. El único modo que permitiría encontrar la clave privada es un ataque por fuerza bruta, vulnerabilidad de que solventa aumentando el mínimo número de bits recomendado para generar las claves. De manera que la publicación de la clave pública **no compromete** de ningún modo la privacidad de la clave privada.

Por supuesto, la destrucción o el robo de la clave privada es el mayor peligro del sistema, puesto que la destrucción haría imposible el descifrado de los mensajes, y el robo permitiría al propietario ilegal de la clave descifrar nuestra información personal.

La criptografía asimétrica sustenta unas diferencias importantes respecto a la criptografía simétrica:

- **No intercambio de la clave privada:** dado que únicamente se comparte la clave pública, la clave privada se mantiene realmente en secreto. Además cada usuario únicamente tiene que almacenar de forma segura su clave privada. Mientras que con la criptografía simétrica los usuarios tienen que almacenar de forma segura una clave para cada comunicación con distintos usuarios.

- **Integridad, autenticidad y no-repudio**: con la infraestructura adecuada, es posible asegurar la confidencialidad además de permitir identificar los remitentes, garantizar el no-repudio del mensaje y asegurar que los datos se han mantenido inalterados durante el envío.
- **Mayor tiempo de proceso:** la criptografía asimétrica hace uso de métodos matemáticos más complejos que la simétrica, por lo tanto, para una misma longitud de clave se requiere un mayor tiempo de cálculo. Además el mensaje cifrado con este método es de mayor tamaño que el mensaje original.

1.2 Infraestructura de clave pública

La infraestructura de clave pública o PKI, de Public Key Infrastructure en inglés, es la combinación de hardware, software, políticas y procedimientos de seguridad que permite vincular las claves públicas con sus respectivas entidades. De manera que una entidad externa en que confían las partes implicadas garantiza que una clave pública pertenece a una entidad.

A continuación se explican las partes implicadas en la PKI.

1.2.1 Firma digital

Uno de los elementos esenciales de la PKI es el proceso de firma digital, ya que tiene un papel muy relevante en el campo de la certificación que se trata en la siguiente sección.

Una firma convencional, es decir, una firma con bolígrafo en un documento tal y como la conocemos, permite autentificar la identidad del autor de este documento, o expresar el consentimiento a su contenido. Una firma convencional es única y va asociada a una persona física.

La firma digital, de manera análoga, también permite autentificar la identidad del autor del documento o mostrar consentimiento a unos términos, pero de forma complementaria puede estar asociada a una entidad no física, como puede ser un software o un servicio. Y no sólo nos permite verificar la identidad del firmante, sino también verificar que el documento **no ha sido modificado** después de haberse realizado la firma.

1.2.1.1 Proceso de creación de una firma digital

Los principios para realizar una firma digital son:

1. Obtener el *hash* de los datos a firmar.
2. Cifrar el *hash* con la clave privada del firmante.
3. Adjuntar la **firma digital** y los **datos**.

Con estos tres pasos se obtienen unos datos o documento firmado tal y como se muestra en la siguiente figura.

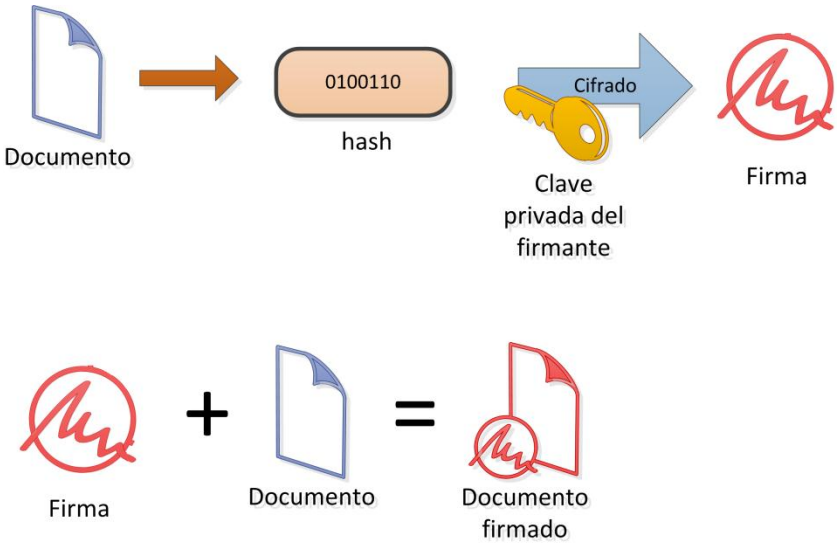


Figura II.5: Proceso de firma digital

1.2.1.2 Proceso de verificación de una firma digital

Para que cualquier entidad pueda verificar una firma digital, es decir, verificar la identidad del firmante y que los datos firmados no han sido modificados después de haberse firmado, hay que seguir los siguientes pasos:

1. Descifrar los datos cifrados con la **clave pública del firmante** para obtener el *hash* original de los datos.
2. Obtener el hash actual de los datos.
3. Comparar ambos *hashes* y verificar que son iguales.

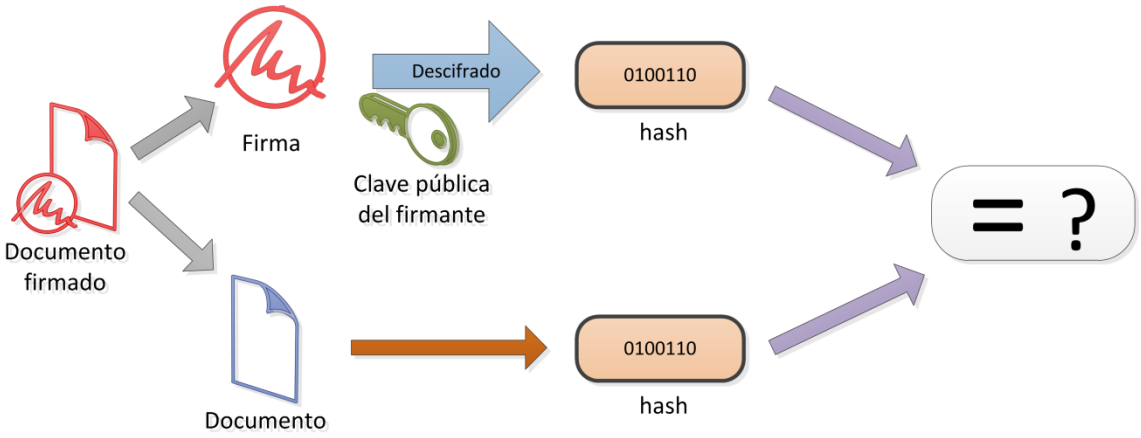


Figura II.6: Proceso de validación de una firma digital

Con este proceso se puede verificar:

- **Integridad**: El documento no ha sido modificado porque el resultado de calcular el hash es el mismo que el hash que ha sido encriptado previamente. Si, por el contrario resultara un hash diferente significa que el documento ha sido modificado.

- **No repudio**: Confirmar la identidad del autor de la firma. Si el descifrado se realiza satisfactoriamente con la clave pública del firmante, por las propiedades de la criptografía asimétrica, implica que la firma ha sido realizada con su clave privada.

1.2.1.3 Firma XML y tipos de firma

Una versión más completa de firma digital es la firma XML o XMLDsig. Ésta permite añadir más información relativa a la pura firma en un documento XML. Entre otros campos encontramos el algoritmo de firmado, el algoritmo de *hash* o información sobre la clave.

Habitualmente cuando se firman datos se usa la XMLDsig en lugar de una firma en formato *raw* o puramente binaria ya que facilita el proceso de validación.

Cuando hablamos de XMLDsig existen tres maneras de generar una firma digital en un documento. El tipo varía en función de la localización de los datos firmados y la firma en el documento XML.

- **Detached Signature**: La firma se separa de los datos firmados en dos ficheros diferentes.

Figura II.7: Firma detached

- **Enveloping Signature:** La firma *enveloping* o envolvente, consiste en crear un documento XML que contenga la firma y los datos que se han firmado.

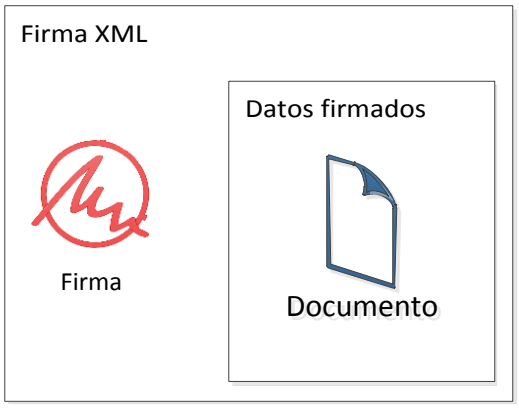


Figura II.8: Firma enveloping

- **Enveloped Signature**: La firma *enveloped* o envuelta, únicamente permite firmar documentos XML y consiste en incluir dentro del propio documento XML firmado el campo <*Signature*> con la firma XML.

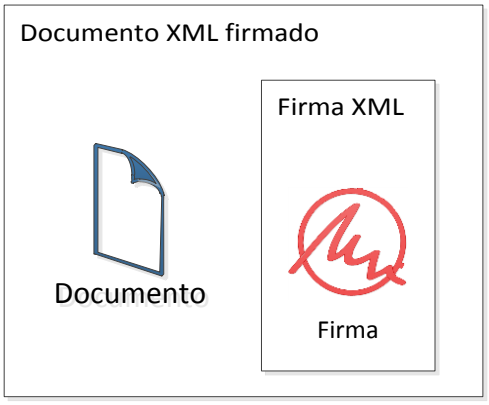


Figura II.9: Firma enveloped

1.2.1.4 Firma PDF

Las firmas de los documentos PDF tienen una serie de características que las diferencian de otros tipos de firma más comunes.

Cuando se firma un PDF, los datos de la firma se incrustan dentro del propio documento, en lugar de estar añadido en un formato de fichero existente o en un fichero aparte. Además esto permite que el lector pueda realizar algunos cambios en el documento sin invalidar la firma.

El certificado del firmante pasa a formar parte de los datos firmados así como información adicional, por ejemplo una representación gráfica de la firma, un sello de tiempo u otros datos relativos al usuario, sistema o aplicación.

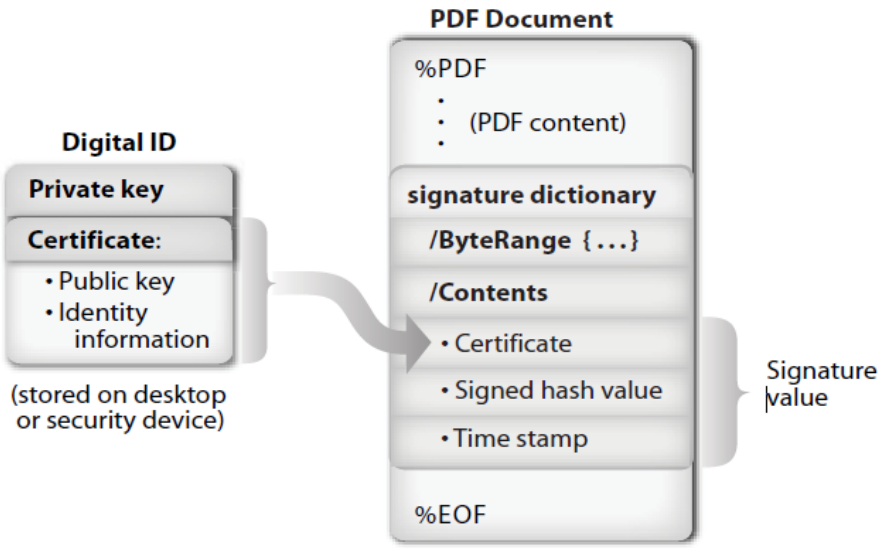


Figura II.10: Estructura de una firma PDF

Todos estos datos se pueden localizar dentro del documento buscando la etiqueta *ByteRange*, un vector de 4 números que se almacena junto la firma. La firma se almacena incrustada en el documento por lo que *ByteRange* indica tanto la primera secuencia (antes de la firma) como la segunda secuencia (después de la firma) de los datos a incluir en el *hash*. La primera secuencia se indica en los dos primeros números del *ByteRange* y la segunda en los dos últimos. El primer número de cada par almacena el punto de inicio de los datos (*offset*) mientras que el segundo almacena su longitud.

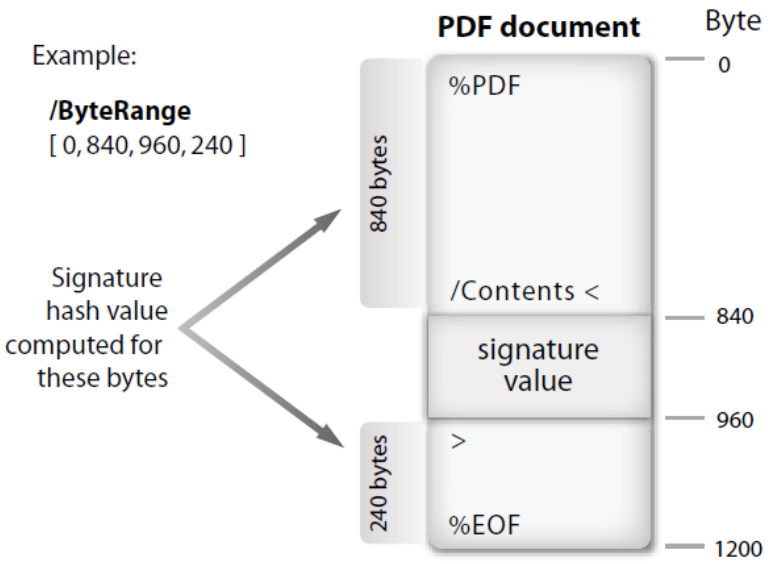


Figura II.11: Ejemplo de firma PDF

El propio valor de la firma se almacena bajo la etiqueta *Contents* entre la primera y la segunda secuencia de datos a firmar. En *Contents* encontramos un objeto PKCS#7[1] codificado en hexadecimal con los datos de la firma, así como el certificado del firmante y los otros datos relacionados.

El hash se computa desde el byte 0 hasta el último byte del fichero, excluyendo los bytes del valor de la firma. El flujo de firma PDF es el siguiente:

1. Se escribe el PDF entero en disco, con un espacio extra para añadir la firma. El vector *ByteRange* contiene valores que contemplan el peor caso posible.

2. Una vez se conoce la localización del valor de la firma en cuanto a *offsets*, se sobrescribe *ByteRange* con los valores correctos. Dado que el *offset* no puede cambiar, los bytes extra se sobrescriben con espacios.

3. Se calcula el *hash* de todo el fichero, utilizando los bytes especificados por *ByteRange*.

4. El *hash* se cifra con la clave privada del firmante y se genera un objeto PKCS#7.

[1] Ver sección 1.2.10 de este capítulo

5. El objeto contenedor de la firma se sitúa en el fichero en disco sobrescribiendo la sección *Contents*. El espacio no utilizado por la firma se sobrescribe con espacios.

Existen otras funcionalidades que caracterizan la firma PDF como la posibilidad de realizar múltiples firmas o certificar los documentos, pero que no se detallarán dado que no son de importancia en el contexto de este proyecto.

1.2.2 Certificado digital

El **certificado digital** es la estructura de datos más importante en la PKI. En secciones previas se ha dicho que un par de claves va asociado a una entidad, siendo la clave pública abiertamente distribuida. Mediante el certificado digital se relaciona la clave pública con la entidad que afirma ser la propietaria.

El certificado es una estructura de datos que contiene el titular de la clave pública, fecha de caducidad y el titular del certificado, entre otros campos. Para verificar la validez del certificado toda la estructura que contiene debe estar firmada por una **tercera parte de confianza**. El titular de la clave pública no puede firmar el certificado ya que entonces cualquiera podría generar certificados con nombres aleatorios. Un certificado que está firmado por el titular de su clave pública se llama certificado **auto-firmado** y no tiene validez excepto si se trata de un **certificado raíz** del que hablaremos más adelante.

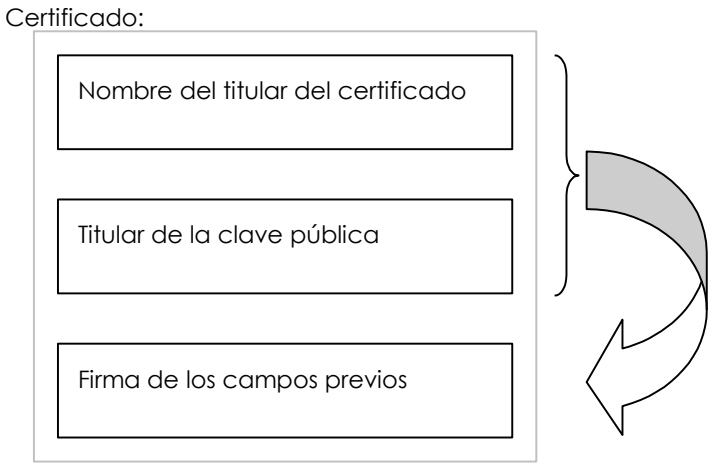


Figura II.12 Estructura básica de un certificado digital

1.2.3 Autoridad de certificación

Una autoridad de certificación o **CA** es la entidad de confianza para todos los participantes que se encarga de firmar y emitir los certificados digitales.

Un certificado firmado por una CA es un certificado válido ya que las CA son de confianza, por lo tanto, su firma tiene validez.

La confianza de la CA se basa en dos factores:

1. La CA sigue suficientes procedimientos de verificación de los datos del titular y verifica que su clave pública sea realmente suya.
2. La clave privada de la CA se mantiene bajo suficientes medidas de seguridad.

Existe una jerarquía de verificación que consiste en que las CA utilizan su clave privada para firmar los certificados de otras CA, por lo que la firmante es superior en la jerarquía. Las CA iniciales en las jerarquías se llaman CA raíz, y sus certificados son auto-firmados.

Dado que la PKI se basa en la confianza si se decide confiar un una determinada CA raíz, se confiará en todas las CA cuyos certificados hayan sido firmados por la CA raíz y así sucesivamente.

1.2.4 Autoridad de registro

El concepto de Autoridad de Certificación se ha tratado anteriormente: es un sistema que dispone de un par de claves que se aplican sobre las peticiones de certificación, emitiendo certificados.

No obstante, los datos que llenan la CA (nombres de titular, su clave pública, etc.) están bajo control. Si la validación de una petición es un proceso automático –la petición de certificación posee una firma verificable con una clave pública incorporada-, el nombre del titular debe verificarse.

La **Autoridad de registro (RA)** comunica las entidades que solicitan certificados con las CAs. Controla la generación de certificados verificando los datos proporcionados por las entidades y emitiendo una petición del certificado a la CA correspondiente, que incluye información del solicitante, su clave pública y un fragmento firmado con su clave privada. Cuando el certificado es generado lo devuelve a la entidad solicitante.

1.2.5 Repositorio de certificados

Los **repositorios de certificados** son almacenes de certificados válidos, es decir, que no han caducado ni han sido revocados, que proporcionan servicios a las entidades para que puedan descargárselos. Suelen estar en directorios X.500[2] accesibles mediante LDAP[3].

1.2.6 Repositorio de listas de certificados revocados

En la sección de certificados digitales se ha mencionado que los certificados tienen fecha de caducidad. Cuando un certificado caduca deja de ser válido, por lo tanto cualquier firma que haya realizado el propietario de certificado caducado deja de ser válida.

Para saber que un certificado ha caducado basta con comprobar su fecha de caducidad, pero existe otro motivo por el cual un certificado tiene que dejar de ser válido y es que la **seguridad de la clave privada haya sido comprometida**.

Cuando la entidad propietaria de un certificado sospecha que su clave privada puede haber sido comprometida, informa a la CA que la ha firmado para que incluya el certificado en su **repositorio de listas de certificados revocados** o **CRL**.

Una CRL contiene los números de serie de los certificados que ya no son válidos por motivos ajenos a su fecha de caducidad. Las listas son emitidas por las CA periódicamente y firmadas por estas para asegurar su validez.

Como el tiempo entre la revocación del certificado y la emisión de una nueva lista de certificados revocados no es inmediato, las operaciones de verificación de firmas digitales no se pueden realizar hasta pasado un período de gracia desde que fue realizada la firma. Así se asegura que el certificado estaba vigente cuando se realizó dicha firma. Los puntos de distribución de CRL se llaman CRL Distribution Point, usualmente están implementados como los repositorios de certificados

Hay una alternativa para conocer el estado de un certificado, el **protocolo OCSP**[4]. Éste permite conocer su estado en línea sin tener que descargar toda la

---

[2] Conjunto de estándares de redes de ordenadores sobre servicios de directorio

[3] Lightweight Directory Access Protocol

[4] Online Certificate Status Protocol

lista de revocación, de modo que la respuesta es más rápida y más actualizada. La entidad que responde a esas peticiones es la Autoridad de validación.

1.2.7 Autoridad de validación

La **Autoridad de Validación (VA)** es una entidad a la que la CA autoriza para que proporcione, de forma online y mediante el protocolo **OCSP**, un servicio de información instantáneo sobre el estado de validez de los certificados digitales. De esta forma se le evita al cliente la necesidad de descargar toda la lista y se proporciona una información realmente actualizada.

1.2.8 Autoridad de sellado de tiempo

La **Autoridad de sellado de tiempo (TSA)** es la autoridad en la que confían las partes implicadas en una comunicación que proporciona certeza sobre la existencia de unos datos en un momento concreto del tiempo.

Esta autoridad es la base del no repudio al añadir el tiempo como referencia. La TSA firma el hash de los datos y la referencia de tiempo con su clave privada para garantizar su existencia. Se puede considerar a la TSA el equivalente informático de los notarios.

1.2.9 Entidad final

Una **entidad final (EE)** es una persona física o un software que posee un certificado con su par de claves, la pública y la privada, que emplea en utilizar los servicios de la PKI (firma, verificación, cifrado, etc.).

1.2.10 Estándares criptográficos

En criptografía existen una serie de estándares llamados PKCS, del inglés *Public-Key Cryptography Standards*, que hacen posible el intercambio seguro de información a través de Internet mediante la infraestructura de clave pública (PKI). Estos estándares incluyen cifrado RSA, cifrado basado en contraseña, sintaxis de certificados extendida y sintaxis de mensajes criptográficos. Fueron desarrollados por *RSA Laboratories* en cooperación con un consorcio que incluía las compañías Apple, Microsoft, DEC, Lotus, SUN y MIT.

Existen un total de 15 PKCS diferentes (algunos de ellos todavía en desarrollo) pero únicamente vamos a definir el conjunto relevante para este proyecto.

Estándar	Versión	Nombre	Comentarios
PKCS#1	2.1	Estándar criptográfico RSA	Define las propiedades matemáticas y el formato de las claves públicas y privadas RSA, así como los algoritmos básicos que realizan el cifrado y descifrado RSA y que producen y verifican firmas.
PKCS#7	1.5	Estándar sobre la sintaxis del mensaje criptográfico (CMS)	Usado para firmar y/o cifrar mensajes bajo PKI. Usado también para la diseminación de certificados (por ejemplo, como repuesta a un mensaje PKCS#10).
PKCS#10	1.7	Estándar de solicitud de certificación	Formato de los mensajes enviados a una Autoridad de certificación para solicitar la certificación de una clave pública.

1.3 Métodos de autenticación

La autenticación es el acto de confirmar que algo es auténtico, en el contexto de este proyecto sería confirmar que la **identidad** del usuario es auténtica.

Existen diferentes métodos para confirmar que una persona afirma ser quien dice ser y se pueden basar en los siguientes factores:

- Algo **conocido**: por ejemplo una contraseña, que es sin duda el método de autenticación más popular, dada su **simplicidad** ya que el usuario únicamente tiene que recordar la secuencia de datos que componen su contraseña.

- Algo **poseído**: este método de autenticación consiste en poseer algo único y que sólo puedes tener tú, por ejemplo una tarjeta o un *token* numérico que cambia según demanda. La **autenticación en dos pasos** utiliza este factor de autenticación. Consiste en después de confirmar la contraseña del usuario, solicitar un el valor numérico de un *token* de autenticación que está en su poder.

- Algo **físico**: también llamada autenticación por biometría, consiste en verificar alguna característica biológica, como puede ser la sangre, la retina, las huellas dactilares, etc.

En este proyecto se mencionarán los métodos de autenticación de **usuario y contraseña** y la **autenticación en dos pasos por OTP**[5].

1.3.1 Autenticación mediante Proveedor de Identidad

Un proveedor de identidad (IdP de las siglas en inglés *Identity Provider*) es un servicio externo que permite autenticar a un usuario en otro servicio sin necesidad de que este cree unas credenciales nuevas.

Por ejemplo, un usuario está registrado en un servicio A y quiere registrarse en un servicio B sin tener que crear unas credenciales nuevas. El servicio A actúa de IdP autenticando el usuario en el servicio B.

Los IdP utilizan protocolos como OpenID, SAML u OAuth 2.0.

[5] One Time Password, una contraseña de un solo uso

2 OAuth 2.0

OAuth 2.0 es un protocolo de autorización que permite a aplicaciones de terceros obtener acceso limitado a un servicio HTTP, tanto en nombre del propietario del recurso (*resource owner*) mediante una interacción de aprobación entre el *resource owner*y el servicio HTTP como dando permisos a una aplicación de terceros para obtener acceso en su nombre.

El RFC del protocolo OAuth 2.0 fue aprobado en octubre de 2012 dejando obsoleto su predecesor, OAuth 1.0 descrito en el RFC 5849.

2.1 Resumen del funcionamiento

Antes de explicar el flujo del protocolo, describiremos los roles que intervienen.

Resource owner: Una entidad capaz de dar acceso a un recurso protegido. Cuando el *resource owner* es una persona, se le refiere como usuario final o *end-user*.

Resource server: El servidor que almacena los recursos protegidos, es capaz de aceptar y responder a peticiones de acceso a los recursos protegidos utilizando tokens de acceso o *access tokens*.

Client: Una aplicación haciendo peticiones a recursos protegidos en nombre del *resource owner* y con su autorización. El término "client" o "cliente" no implica ninguna característica particular de implementación (por ejemplo, si la aplicación se ejecuta en un servidor, escritorio, u otro dispositivo).

Authorization server: El servidor que genera los *access tokens* al cliente después de autenticarse correctamente al *resource owner* y obtener autorización.

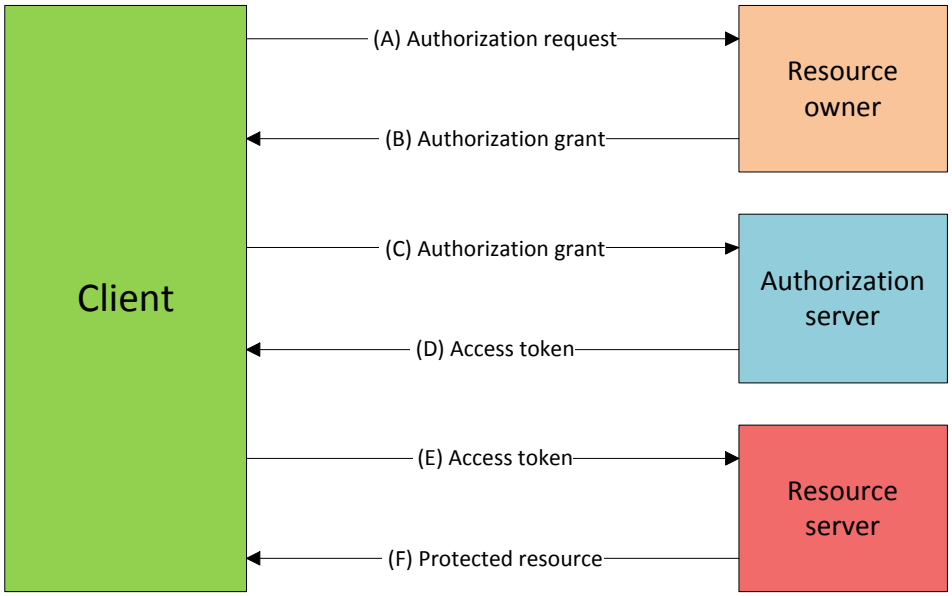


Figura II.13: Flujo OAuth 2.0

A continuación se detalla la interacción entre los 4 roles descritos anteriormente.

(A) El *client* pide autorización al *resource owner*. La petición de autorización se puede hacer directamente al *resource owner* (tal y como se muestra), o indirectamente a través del *authorization server* como intermediario.

(B) El cliente recibe una concesión de autorización o *authorization grant*, que es una credencial que representa la autorización del *resource owner*, expresada usando uno de las 4 concesiones diferentes que se definen en el protocolo OAuth 2.0. El tipo de concesión depende del método usado por el *client* para pedir la autorización y de los tipos soportados por el *authorization server*.

(C) El cliente pide un *access token* autenticándose con el *authorization server* y presentando el *authorization grant*.

(D) El *authorization server* autentica al *client* y valida el *authorization grant*, y si es válido, emite un *access token*.

(E) El *client* pide el recurso protegido del *resource server* y se autentica presentando el *access token*.

(F) El *resource server* valida el *access token*, y si es válido, sirve la petición.

2.2 Tipos de authorization grant

Un *authorization grant* es una credencial que representa la autorización del *resource owner* y se usa por el *client* para obtener un *access token*. Existen cuatro

tipos diferentes de *authorization grant*: *authorization code*, *implicit*, *resource owner password credentials* y *client credentials*. A continuación se describe cada tipo.

2.2.1 Authorization Code

El *authorization code* se obtiene usando un servidor de autorización como intermediario entre el *client* y el *resource owner*. En lugar de hacer la petición de autorización directamente al *resource owner*, el *client* redirige al *resource owner* al servidor de autorización, que a su vez redirige al *resource owner* de vuelta al *client* con el *authorization code* o código de autorización.

Antes de dirigir al *resource owner* de vuelta al *client* con el *authorization code* , el *authorization server* autentica al *resource owner* y obtiene autorización. Las credenciales del *resource owner* nunca se comparten con el *client*, dado que el *resource owner* solamente se autentica con el *authorization server*.

El siguiente diagrama explicado detalla el flujo:

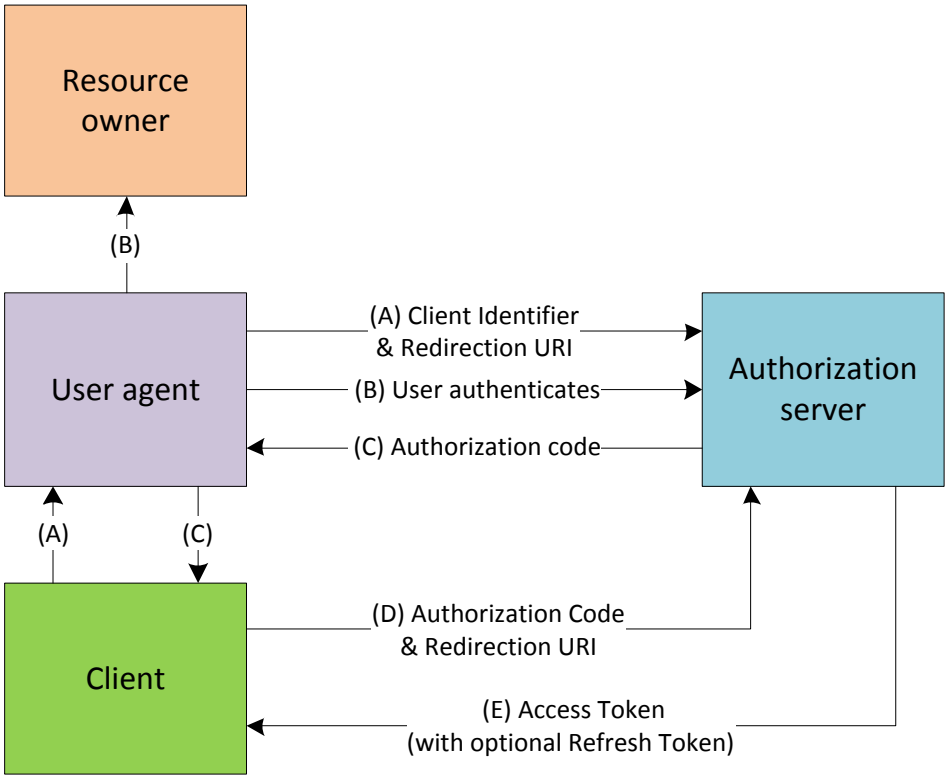


Figura II.14: *Authorization code*

(A) El cliente inicia el flujo redireccionando el *user-agent* del *resource owner* al punto de redirección (típicamente un formulario). El cliente introduce su identificador, su ámbito de acceso, estado y una URI de redirección a la

que el *authorization server* mandará al *user-agent* una vez el acceso finalice (satisfactoriamente o erróneamente).

(B) El *authorization server* autentica al *resource owner* (a través del *user-agent*) y establece si el *resource owner* concede o deniega la petición de acceso del cliente.

(C) Asumiendo que el *resource owner* concede el acceso, el *authorization server* redirecciona el *user-agent* de vuelta al cliente usando la URI de redirección proporcionada anteriormente (en la petición o durante el registro del cliente). La URI de redirección incluye un código de autorización y cualquier estado local proporcionado por el cliente anteriormente.

(D) El cliente pide un *access token* al *authorization server* incluyendo el código de autorización en la petición, obtenido en el paso anterior. Cuando hace la petición, el cliente se autentica con el *authorization server*. En cliente incluye la URI de redirección usada para obtener el código de autorización, con propósitos de verificación.

(E) El *authorization server* autentica al cliente, valida el código de autorización y se asegura que la URI de redirección recibida coincide con la usada para redirigir al cliente en el paso (C). Si todo es correcto, el *authorization server* responde con un *access token* y, opcionalmente, con un *refresh token*.

El *authorization code* ofrece unos añadidos importantes de seguridad, tales como la capacidad de autenticar al *client*, o la transmisión del *access token* directamente al *client* sin pasar por el *user-agent* del *resource owner* y potencialmente exponerlo a otros, incluyendo al *resource owner*.

2.2.2 Implicit

La concesión implícita o *implicit grant* es una versión simplificada del flujo de *authorization code* para clientes implementados en un navegador usando un lenguaje de *scripting* tal como *javascript*. En el fujo de *implicit grant*, en lugar de emitir un *authorization code* al *client*, se le emite directamente un *access token*. La concesión es implícita dado que no se emiten credenciales intermedias como el *authorization code*.

Cuando se emite un *access token* durante la concesión implícita, el *authorization server* no autentica al *client*. En algunos casos, la identidad del *client* puede ser verificada mediante la URI de redirección usada para entregar el *access token* al *client*. El *access token* puede ser que se exponga al *resource owner* o a otras aplicaciones con acceso al *user-agent* del *resource owner*.

La concesión implícita mejora la respuesta y la eficiencia de algunos clientes (por ejemplo los implementados en una aplicación de navegador web), dado que reduce el número de mensajes requeridos para obtener el *access token*. Sin embargo, esta conveniencia debería ser valorada ante las implicaciones de seguridad que conlleva unas *implicit grants*, especialmente cuando la concesión de *authorization code* está disponible.

2.2.3 Resource owner password credentials

Este tipo de autorización por credenciales (por ejemplo nombre de usuario y contraseña) se puede usar directamente como una concesión de autorización para obtener el *access token*. Las crendenciales solamente deberían ser utilizadas cuando hay un alto grado de confianza entre el *resource owner* y el *client* (por ejemplo, el *client* es parte del sistema operativo del dispositivo o una aplicación con altos privilegios), y cuando otros tipos de concesiones de autorización no están disponibles (como el *authorization code*).

Aunque este tipo de concesión requiere acceso directo del cliente a las credenciales del *resource owner*, estas son usadas para una única petición e intercambiadas por un *access token*. Esta concesión puede eliminar la necesidad del cliente de almacenar las credenciales del *resource owner* para un futuro uso, intercambiando las credenciales por un *access token* de larga duración o un *refresh token*.

2.2.4 Client credentials

Las credenciales de usuario o *client credentials* (u otras formas de autenticación de cliente) se pueden usar como una concesión de autorización cuando el contexto de autorización está limitado a los recursos protegidos que están bajo control del *client*, o a los recursos protegidos previamente acordados con el *authorization server*. Las credenciales de usuario, típicamente se usan como concesión de autorización cuando el *client* actúa en su propio nombre (el *client* es también el *resource owner*) o si solicita acceso a recursos protegidos basado en una autorización previamente arreglada con el *authorization server*.

2.3 Ventajas respecto a otras opciones

OAuth 2.0 como protocolo de autenticación y autorización es una muy buena opción en cuanto a servicios de provisión de identidad. Sin embargo, se puede pensar que existen alternativas igual de buenas para esta tarea. Un ejemplo de alternativa podría ser el protocolo OpenID.

2.3.1 OAuth 2.0 vs OpenID

OpenID es un protocolo abierto de autenticación. Cuando se usa para autenticar al usuario en un servicio, este tiene que dar sus credenciales al proveedor de identidad que implementa OpenID para que, de forma similar a OAuth 2.0, este ceda información de la identidad del usuario al consumidor y así proceder a la autenticación en el servicio.

Al ser un protocolo abierto descentralizado nadie posee el control sobre éste. Cualquier servicio con control de autenticación de usuarios puede implementar OpenID para que sea usado como proveedor de identidad en otro servicio.

Estos puntos que se han mencionado son comunes con OAuth 2.0, sin embargo, con OpenID no se puede ofrecer un acceso a los recursos del proveedor, mientras que con OAuth 2.0, siempre que el usuario lo haya autorizado y el *token* de acceso tenga validez, una aplicación de terceros podrá acceder a esos recursos. Con OpenID únicamente se ofrece acceso a recursos de identidad básicos (nombre, *e-mail*, etc).

2.3.2 OAuth 2.0 vs SAML

SAML, o *Security Token Markup Language*, es un *framework* basado en XML que permite compartir información de identidad y seguridad entre diferentes dominios de seguridad. Su especificación, a pesar de estar diseñada en un principio para proveer *single sign-on* entre diferentes navegadores, también lo fue para ser modular y extensible y así facilitar su uso en otros contextos.

Una aserción SAML es una construcción fundamental de SAML que se adopta frecuentemente para su uso en otros protocolos y especificaciones. Normalmente, estas aserciones son emitidas por un IdP y consumidas por un *service provider*, que delega en su contenido para identificar el sujeto de la aserción.

Mediante la especificación *OAuth 2.0 Assertion Profile* podemos utilizar aserciones SAML para obtener los *tokens* OAuth. Esta especificación define una extensión de *grand type* que usa un *SAML 2.0 Bearer Assertion* para solicitar un *token* OAuth 2.0 y a su vez para usarlo de credenciales de usuario.

Esto permite ampliar el protocolo OAuth 2.0 con un método que puede escalar sus posibilidades y ofrecer alternativas a los tradicionales métodos de *authorization grant* comentados en la sección 2.2.

3 Servicios web

Mediante un servicio web se pueden intercambiar datos entre aplicaciones. Se compone por un conjunto de estándares y protocoles que permiten que distintas aplicaciones, desarrolladas en diferentes plataformas y lenguajes puedan hacer uso de los servicios web para comunicarse a través de redes, como Internet. Un ejemplo de estándar de servicios web es SOAP[6].

4 Cloud Computing

El *Cloud Computing*, o computación en la nube, es un paradigma que permite ofrecer servicios de computación a través de la red El nombre proviene del uso en diagramas de sistema de símbolos con forma de nube como una abstracción de la compleja estructura que contiene.

El *Cloud Computing* en sí es un nombre muy genérico y engloba un conjunto de modelos públicos y privados. En este conjunto podemos encontrar los tres modelos principales (en negrita):

- **Infraestructura como un servicio (IaaS)**
- **Plataforma como un servicio (PaaS)**
- **Software como un servicio (SaaS)**
- Red como un servicio (NaaS)
- Almacenamiento como un servicio (STaaS)
- Seguridad como un servicio (SECaaS)
- Datos como un servicio (DaaS)
- Bases de datos como un servicio (DBaaS)
- Entornos de prueba como un servicio (TEaaS)
- Virtualizaciones de escritorio
- API como un servicio (APIaaS)
- *Backend* como un servicio (BaaS)

Como se aprecia, cualquier modelo de un entorno de ejecución local que se pueda extrapolar a un servicio ofrecido en la red puede formar parte del *Cloud Computing*. De hecho, el *Cloud Computing* no es ninguna tecnología nueva, sino

[6] Simple Object Access Protocol.

una **unión de muchas tecnologías** ya conocidas que proveen de un servicio global y escalable en función de la demanda de los usuarios.

4.1 Modelos principales

Los principales modelos que se encuentran dentro de la computación en la nube, de los listados anteriormente, son la **infraestructura como un servicio** la **plataforma como un servicio** y el **software como un servicio**.

- **IaaS**: Cuando decimos infraestructura como un servicio nos referimos a la capacidad dada al consumidor para proveerle de **proceso**, **almacenamiento**, **redes** y otros **recursos computacionales** fundamentales, donde éste puede desplegar y ejecutar cualquier tipo de software, incluyendo sistemas operativos y aplicaciones. El consumidor no controla la capa inferior de la infraestructura en la nube, pero tiene control sobre sistemas operativos, almacenamiento y diferentes aplicaciones desplegadas. También puede tener un control limitado sobre elementos de la red (por ejemplo firewalls).

- **PaaS:** Cuando al consumidor se le ofrece una plataforma como un servicio, se le ofrece la posibilidad de **desplegar** en la infraestructura en la nube cualquier tipo de **aplicación** (sea creada o adquirida) que usen lenguajes de programación, librerías, servicios o herramientas soportadas por el proveedor. El consumidor no controla la infraestructura subyacente en la nube, incluyendo redes, servidores, sistemas operativos o almacenamiento, sin embargo tiene control sobre las aplicaciones desplegadas y sobre algunos parámetros de configuración del entorno de ejecución.

- **SaaS:** El modelo de más alto nivel da al consumidor la capacidad de usar las **aplicaciones del proveedor** que se están ejecutando en la infraestructura en la nube. A estas aplicaciones se puede acceder desde varios dispositivos de cliente, sean con una interfaz ligera, como un navegador web (un ejemplo sería el web-mail), o con una interfaz de programa. El consumidor no controla la infraestructura subyacente en la nube, incluyendo redes, servidores, sistemas operativos, almacenamiento o ni siquiera características individuales de la aplicación, con la posible

excepción de parámetros de aplicaciones específicas para algunos usuarios.

En la siguiente figura vemos un esquema representativo de estos tres modelos principales. Podemos apreciar cómo, siguiendo una jerarquía, los modelos más superiores ofrecen servicios de más alto nivel, mientras que los inferiores ofrecen los de más bajo nivel.

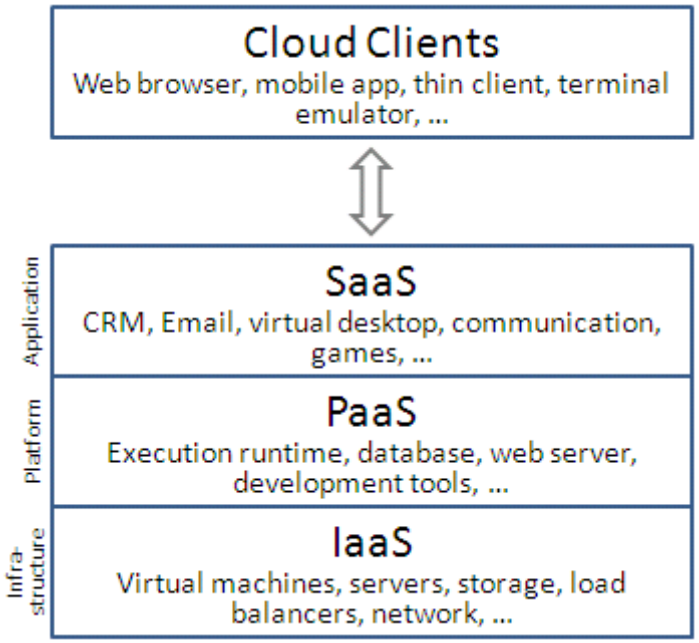


Figura II.15: Jerarquía de los modelos de *Cloud Computing*

4.2 Almacenamiento de ficheros en la nube

Cuando hablamos de almacenar ficheros en la nube (*Cloud Storage*, en inglés) nos referimos a utilizar uno de los servicios ofrecidos por el *Cloud Computing*, cuya principal funcionalidad es almacenar y gestionar datos bajo demanda... Los datos se alojan en espacios de almacenamiento virtualizados y por lo general están alojados por terceros.

En este proyecto se decidió utilizar esta funcionalidad del *Cloud Computing* para almacenar los documentos PDF que tienen que tratarse a través de una aplicación de un dispositivo móvil. Si desde el dispositivo únicamente se ven los títulos de los documentos en cuestión, esto reduce el tráfico de datos y externaliza la gestión de almacenaje, de manera que el usuario puede ver sus documentos desde cualquier aplicación que implemente este servicio web.

4.2.1 Clasificación dentro del Cloud Computing

Dentro de todos los modelos del *Cloud Computing* el *Cloud Storage* podría encontrase en dos modelos diferentes: **SaaS** o **IaaS**.

Esto depende del punto de vista con el que se mire al servicio. Cuando hablamos de simple almacenamiento de datos, esto es, almacenar, recuperar, eliminar y modificar, puede encajar perfectamente dentro del modelo de *infraestructura*, es decir, **IaaS**. Podemos analizar la definición dada anteriormente para justificar esta afirmación donde se especifica que un IaaS ofrece servicios como el almacenamiento.

Sin embargo un servicio de almacenamiento podría considerarse una aplicación que corre sobre un sistema operativo que a su vez utiliza un sistema de archivos determinado. En este caso el *Cloud Storage* podría ubicarse dentro de *software*, es decir, **SaaS**.

Se pueden dar ejemplos de servicios reales donde el *Cloud Storage* es ubicado en ambos modelos. En el caso de **IaaS** podemos encontrar *Amazon Web Services* o *Rackspace*, ambos ofrecen una infraestructura como un servicio, entre la cual encontramos el servicio del almacenamiento, pero también ofrecen muchos otros elementos con los que poder desplegar cualquier tipo de software. En el caso de **SaaS** podemos encontrar *Box*, *Dropbox* o *Google Drive*, donde claramente se ofrece únicamente una aplicación que permite el almacenamiento de archivos, pero aparte de esta funcionalidad que se puede configurar de manera muy limitada no se puede controlar nada relativo a la infraestructura sobre la que se ejecuta. Estos servicios ofrecen al usuario, aparte del puro almacenamiento, un acceso a sus ficheros a través de una interfaz de aplicación. Esta interfaz puede ser web, de programa o incluso el mismo sistema operativo, ya que, en muchos de ellos, se ofrece la capacidad de sincronizar determinados directorios del sistema operativo con el servicio de la nube. De esta manera, cualquier fichero que se encuentre dentro del directorio se subirá automáticamente al *Cloud Storage*.

Para el caso que nos ocupa se encuentra más interesante esta última opción, ya que es más limitada y más simple, porqué únicamente se necesita controlar un número no muy grande de archivos por usuario y con la máxima simplicidad posible sin la necesidad de desplegar ningún tipo de software sobre el *cloud*. También resulta interesante la posibilidad que los usuarios accedan a sus ficheros a través de la aplicación web del servicio. Analizaremos uno de los principales servicios del mercado y que también se usa en este proyecto: **Google Drive**.

4.2.2 Google Drive

Google Drive es el servicio de almacenamiento en línea de la empresa **Google**. Fue establecido el 24 de abril de 2012 como reemplazo a Google Docs. Como servicio principal ofrece 5 GB de almacenamiento gratuito a sus usuarios, ampliables mediante el pago de cuotas.

Ofrece un acceso mediante una interfaz web a todos los archivos y una integración con Google Docs para editar los documentos ofimáticos directamente desde el navegador. También permite una integración con los escritorios de los principales sistemas operativos y un acceso a través de aplicación móvil.

Pone a disposición de los desarrolladores una API pública accesible mediante REST[7] con la que pueden crear aplicaciones que se beneficien del servicio de almacenamiento. Por lo tanto, una aplicación de terceros podrá tener acceso a los documentos de sus usuarios almacenados en Google Drive, siempre que éste dé su autorización.

Las aplicaciones que requieran autorización de sus usuarios para acceder a los documentos que tienen almacenados en Google Drive deberán obtenerla mediante el protocolo OAuth 2.0 que se ha explicado en este documento. Para ello Google ofrece una API en múltiples lenguajes de programación, incluyendo Java, con métodos que facilitan esta tarea.

5 Tecnologías de Safelayer

En este apartado se definen las principales tecnologías de Safelayer implicadas en el desarrollo del proyecto.

5.1 TrustedX

TrustedX es una plataforma basada en SOA[8], desarrollada por la empresa Safelayer Secure Communications, S.A cuyo objetivo es simplificar el uso de servicios de confianza basados en Infraestructuras de Clave Pública para firma

[7] Transferencia de Estado Representacional, en inglés Representational State Transfer, es una técnica de arquitectura de software para sistemas hipermedia distribuidos. En la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML y HTTP.

[8] Del inglés, Service Oriented Architecture.

electrónica, protección de datos y cualquier tipo de gestión de identidad electrónica.

Dada su arquitectura orientada a servicios la integración con otras plataformas o aplicaciones resulta muy sencilla, reduciendo la complejidad que hasta la fecha suponía el dotar a cualquier aplicación de mecanismos de seguridad y *PKI*. Las funcionalidades que incluye la plataforma son las siguientes:

• **Firma electrónica:** permite la verificación y generación de firmas. Se reconocen diferentes prestadores de certificación y se permite generar y custodiar evidencias electrónicas necesarias para que las firmas puedan ser verificadas a lo largo del tiempo.

• **Protección de datos:** permite la protección de datos y su custodia garantizando el mantenimiento a lo largo del tiempo y el acceso a éstos por parte de entidades autorizadas.

• **Gestión de claves:** permite registrar, revocar, consultar y verificar las claves de las entidades.

• **Autenticación, autorización y control de acceso:** a través de un servicio común, se permite la autenticación, autorización y control de acceso de las entidades registradas haciendo posible el control de acceso único en toda la plataforma.

• **Gestión de objetos y entidades:** se proporciona un modelo de información uniforme basado en XML para todos los objetos y entidades de la plataforma, enmascarando totalmente formatos, localizaciones, fuentes de información, etc. Se ofrecen funciones de registro, consulta y modificación de la información sobre entidades.

• **Auditoria y accounting:** se gestiona de forma centralizada y uniforme toda la información de logs de todos los servicios así como la información de uso y/o consumo de los mismos.

5.2 SmartWrapper

SmartWrapper es una API desarrollada por Safelayer sobre Axis para crear aplicaciones cliente que utilicen los servicios de *TrustedX*. *SmartWrapper* permite generar aplicaciones Java de manera más sencilla ya que evita la complejidad

de programar utilizando directamente Axis[9]. Aun así es el acceso a las estructuras Axis para crear llamadas avanzadas.

Para invocar un servicio de *TrustedX* y evaluar su respuesta es necesario crear una instancia de la clase *Request* correspondiente al servicio que se requiera, se utilizan las funcionalidades para construir una petición válida (añadir la cabecera de autenticación, política utilizada, datos que se van a tratar, etc.) y finalmente se envía la petición que devolverá un objeto de tipo *Response* con los resultados de la operación.

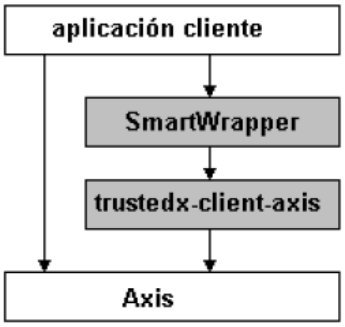


Figura II.16: Jerarquía de *SmartWrapper*

SmartWrapper supone una herramienta muy útil para integrar los servicios de *TrustedX* en aplicaciones Java.

[9] Apache Axis, implementación libre del protocol SOAP

III - INTRODUCCIÓN Y ANÁLISIS DE LAS CROSS-PLATFORM TOOLS

El desarrollo para aplicaciones móviles para *Smartphones* es un sector en alza. Sin embargo, se ve sometido a una alta fragmentación debido a la variedad de dispositivos disponibles en el mercado. Liderados por el duopolio Google – Apple, entre ellos encontramos Android, iOS (iPod, iPhone e iPad), Blackberry, Symbian, Windows Phone 7, y otros sistemas más minoritarios como Samsung Bada.

Esta fragmentación causa una serie de problemas tanto a los desarrolladores como a las empresas productoras de aplicaciones para estos dispositivos.

Muchas plataformas distintas implican muchos entornos de desarrollo distintos y, a menudo, muchos lenguajes de programación distintos. Por lo tanto, un equipo especializado en desarrollar para una plataforma, difícilmente se podrá especializar en el desarrollo de aplicaciones para otra plataforma distinta, consecuentemente, una empresa necesitará varios equipos especializados en el desarrollo orientado a cada plataforma en concreto. Comúnmente, Android con Java y Eclipse e iOS con Objective-C y un Mac con XCode.

Una empresa que quiera desarrollar aplicaciones multiplataforma se ve afectada por un incremento, tanto temporal (por la necesidad de coordinar equipos compuestos por distintos especialistas) como económico (por el hecho de tener que contar con varios equipos).

Para solventar estos inconvenientes procedentes de la fragmentación del mercado existen las herramientas llamadas Cross-Platform Tools (CPT), herramientas orientadas al desarrollo para múltiples plataformas, permitiendo al desarrollador usar un único entorno y lenguaje de programación y maximizar el reaprovechamiento de código entre plataformas.

Otra de las principales ventajas de estas herramientas es permitir a los desarrolladores alcanzar plataformas a las que no podrían optar de otra manera. Muchas de ellas están basadas en el desarrollo web. Con el alza de HTML5, la combinación de las tecnologías HTML, CSS y JavaScript, se han abierto las puertas al desarrollo de aplicaciones en entorno web. La mayoría de las CPT aprovechan las funcionalidades de HTML5 para permitir al desarrollador web entrar en el mercado de los *Smartphone*.

1 Clasificación

Existen 3 grandes grupos de CPT, cada uno de ellos orientado al desarrollo de un tipo de aplicación por un tipo de desarrollador distinto.

1.1 Web Apps

Son aplicaciones web desarrolladas principalmente con la tecnología HTML5 (existen otros tipos más obsoletos como Flash) a las que se accede mediante el navegador del *Smartphone* y simulan un comportamiento nativo a pesar de no estar descargadas en el dispositivo como tales.

La evolución de los sistemas operativos para móviles avanza rápidamente y HTML5 está lejos de poder ofrecer una experiencia nativa al usuario. Por ejemplo, su API no ofrece soporte para NFC[10] ni notificaciones remotas por Push[11].

Asimismo, los desarrolladores tienen que lidiar con el gran nivel de variación con el que los diferentes navegadores implementan las especificaciones de HTML5.

La siguiente tabla muestra la fragmentación en el soporte de funcionalidades de HTML5 de los navegadores móviles.

[10]Del ingles, Near Field Communication.Tecnología de comunicación inalámbrica de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos recientemente implantada en los *Smartphones*.

[11] Es un estilo de comunicaciones sobre Internet donde la petición de una transacción se origina en el servidor.

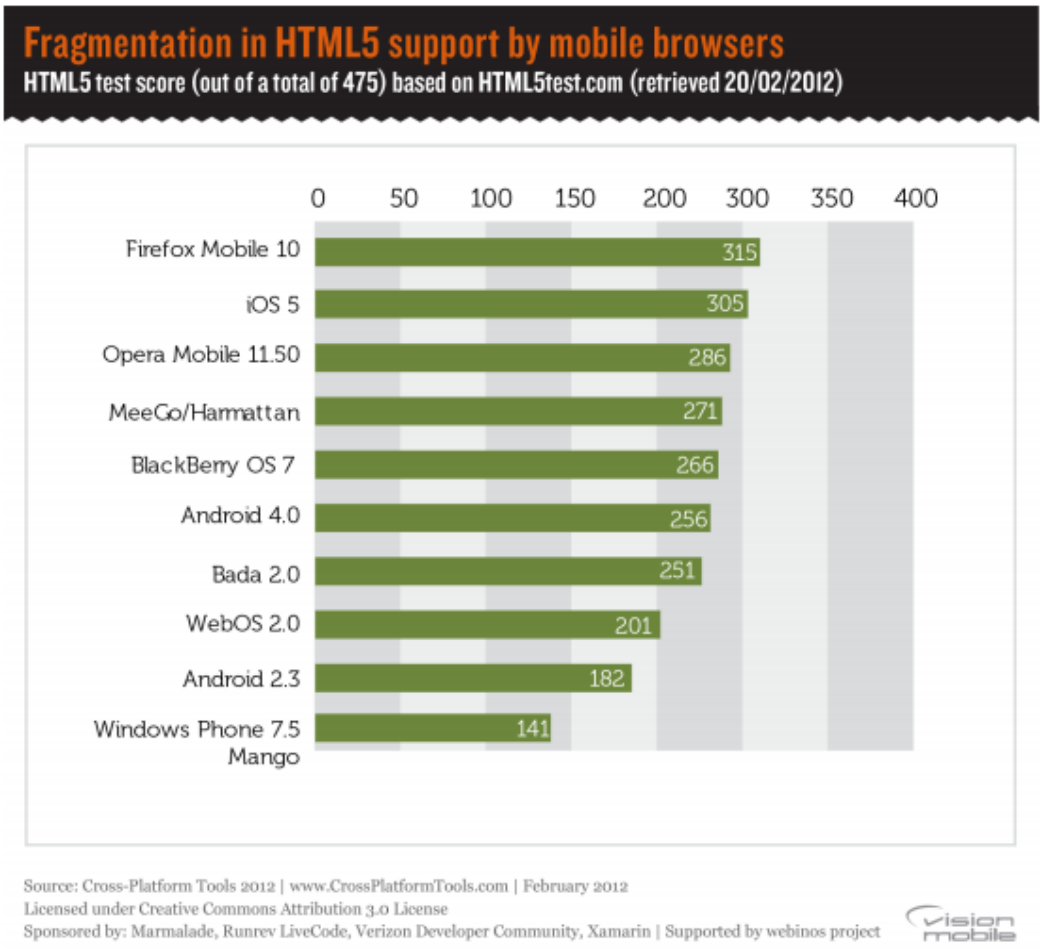


Figura III.1: Fragmentación en el soporte de HTML5 en los navegadores web móviles

Fuente: www.CrossPlatformTools.com

Podemos destacar como, por ejemplo, el navegador integrado en Android 4.0 se encuentra 4 niveles por debajo del de iOS. Por lo tanto el desarrollador no podrá esperar que la misma aplicación, se comporte de la misma forma en un iOS que en un Android y tendrá que adaptar la aplicación para un sistema u otro.

Hay que considerar que creando una aplicación web se descarta la posibilidad de publicarla en cualquiera de los mercados disponibles ("App Store", "Google Play", etc), de manera que tanto la distribución (que tendrá que hacerse a través de las redes sociales, posicionamiento en buscadores, o blogs) como la obtención de beneficios se complica. Este es un factor importante a tener en cuenta ya que generalmente cuando una empresa crea una aplicación querrá también posicionarla en todos los mercados posibles para, así, obtener el mayor número de usuarios y beneficios.

Ejemplos de CPT orientados al desarrollo de Web Apps son:

- Sencha
- JQuery Mobile
- iUI
- Strobe / Sproutcore

1.2 Híbridas

Las aplicaciones híbridas permiten a los desarrolladores web, crear aplicaciones nativas, esto es, que se ejecuten de la misma manera que cualquier otra aplicación nativa y que se puedan publicar de cualquiera de los mercados como otra aplicación más.

Desde el punto de vista del usuario, una aplicación híbrida no se distingue de una aplicación completamente nativa. La descubre y la descarga usando el mercado de aplicaciones nativo del dispositivo y se instala usando el procedimiento común. Una vez instaladas se pueden encontrar junto con las demás aplicaciones y pueden funcionar sin conexión a Internet.

Para el desarrollador, la creación de una aplicación híbrida no se distingue mucho de la creación de una aplicación web ya que la mayor parte del código está compuesto por HTML, JavaScript y CSS. Este código es después incrustado en una aplicación nativa compuesta entera o parcialmente por un una instancia de un navegador web (*WebView*) sin controles de navegación para que la apariencia final sea de aplicación nativa.

El siguiente gráfico muestra la arquitectura de las aplicaciones web junto con la de las aplicaciones híbridas.

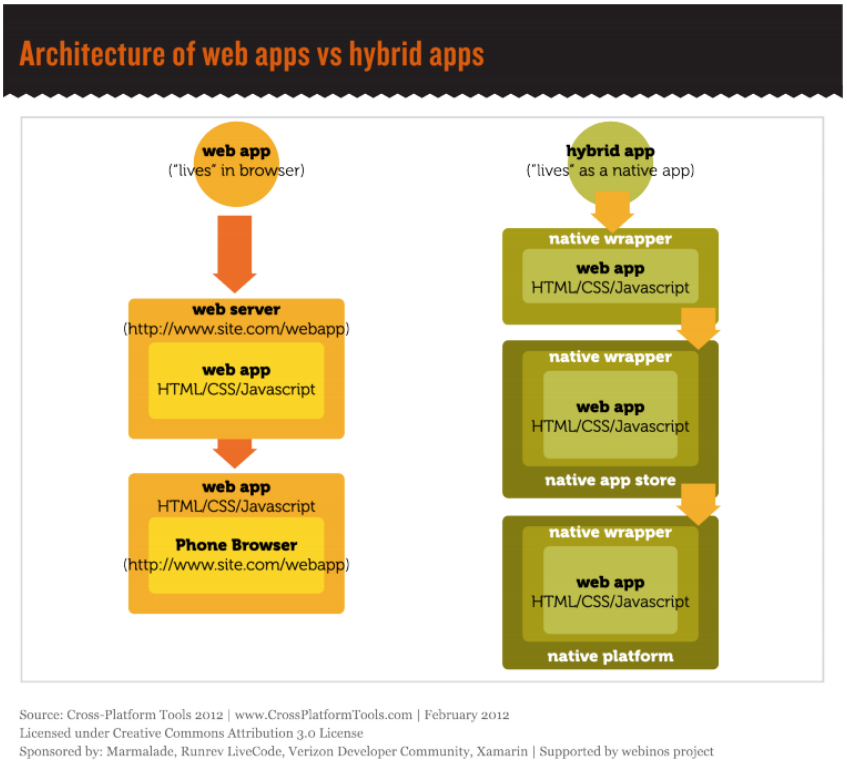


Figura III.2: Arquitectura de las web apps vs las hybrid apps

Fuente: www.CrossPlatformTools.com

De manera que este tipo de aplicaciones combinan la potencia del HTML5 con las mismas opciones de distribución de las aplicaciones nativas. Sin embargo, lo que acaba ejecutando el teléfono no es más que lo mismo que se ejecuta en un navegador por lo que las limitaciones de rendimiento son las mismas.

Muchos desarrolladores que dejan de usar este tipo de CPT lo hacen debido a los problemas de rendimiento y falta de integración con las APIs del dispositivo. Aunque las ventajas a nivel de reutilización de código y simplicidad de programación (para cualquier persona habituada a desarrollar en entorno web) hacen que también sean de las plataformas con más acogida entre los desarrolladores ya que permiten crear aplicaciones multiplataforma de forma rápida y fácil. Por lo tanto, dado que muchos desarrolladores las prueban, pero pocos se quedan con ellas no acaba de haber una consolidación firme.

Entre las principales CPT que permiten crear aplicaciones híbridas podemos encontrar:

- PhoneGap
- BKRender

- Sencha
- Worklight

1.3 Nativas

Este tipo de aplicaciones, al contrario que las anteriores, están orientadas a desarrolladores de software (familiarizados con los lenguajes de alto nivel cómo C#, Java o de más bajo nivel cómo C++) en lugar de estarlo a los desarrolladores web.

Dentro de este subconjunto encontramos dos tipos de CPTs distintos: **runtimes** y **traductores de código fuente**.

1.4 Runtimes

Los *runtimes* consisten en un entorno de ejecución situado entre el desarrollador y la plataforma nativa generando una abstracción entre la aplicación y el dispositivo.

Estos varían en tamaño y complejidad y pueden ejecutar el código mediante métodos diferentes – virtualización, intepretación, compilación *just-in-time* o compilación *ahead-of-time*.

Ejemplos de CPTs de tipo runtime son:

- J2ME
- Unity
- Xamarin (MonoTouch y Mono for Android)
- Appcelerator

1.5 Traductores de código fuente

Estas soluciones se basan, como el nombre indica, en traducir el código fuente a un código intermediario (como puede ser C++, Objective-C, o JavaScript) o directamente a código máquina de bajo nivel, es decir, ensamblador.

Son soluciones frecuentemente usadas a la par con un runtime y orientadas a desarrolladores software avanzados que requieren exprimir al máximo las capacidades del dispositivo.

Entre los CPTs de tipo runtime podemos encontrar:

- MoSync
- Eqela
- Marmalade

2 Evaluación de Cross-Platform Tools

En esta sección se analizarán unas cuantas aplicaciones creadas con una selección de diferentes CPT. Principalmente los atributos a analizar serán el rendimiento, la interfaz gráfica de usuario y las diferencias entre versiones de la aplicación para distintas plataformas, concretamente iPhone y Android.

Los dispositivos usados para las pruebas son un iPhone 4 con iOS 5 y un HTC Magic con Android 2.2.1. Hay que tener en cuenta que las especificaciones técnicas de estos dispositivos distan bastante entre ellas, siendo el iPhone 4 mucho más potente que el HTC Magic. Debido a esto y por razones obvias, el rendimiento de las aplicaciones en HTC Magic será considerablemente más lento. Más allá de ser un simple hándicap en las pruebas es otra consideración a tener en cuenta cuando se desarrolla empleando CPT, ya que si las aplicaciones tienen que ejecutar JavaScript en lugar del código nativo del dispositivo van a resultar siendo considerablemente más lentas si este no es lo suficientemente potente.

Los CPTs investigados son los 3 más usados según un estudio de Vision Mobile:

- PhoneGap
- Sencha Touch
- Xamarin (MonoTouch y Mono for Android)

2.1 Adobe PhoneGap

PhoneGap es una de las plataformas más acogidas entre los desarrolladores así como de las más abandonadas. Basa su funcionamiento en la creación de aplicaciones **híbridas** con **HTML5**. Permite crear aplicaciones utilizando Javascript, HTML5 y CSS3 para dispositivos móviles, en lugar de lenguajes específicos de dispositivo como Objective-C, siendo estas capaces de acceder a las API nativas del dispositivo. El resultado, como ya se ha dicho es de una aplicación híbrida, esto es, no es ni puramente nativa porqué las vistas están generadas mediante *webviews*, ni puramente *webapps*, porqué está empaquetada como una aplicación más, con sus opciones de distribución y tiene acceso a las API nativas (contactos, cámara, sensores, etc.).

A continuación se realiza un análisis del comportamiento general de aplicaciones creadas con este CPT así como las interfaces gráficas de cada una.

2.1.1 Análisis

En este apartado se analizan algunas aplicaciones realizadas utilizando el *framework* PhoneGap.

La mayoría de las aplicaciones desarrolladas con PhoneGap se caracterizan, al componerse en su totalidad de código web (HTML, JavaScript y CSS), por tener el mismo aspecto en su interfaz gráfica tanto en Android cómo iOS.

Por consiguiente, un usuario que ejecute una aplicación en un dispositivo Android e iOS, no debería notar diferencia alguna. A pesar de ello, sí se aprecian diferencias dado que este tipo de aplicaciones híbridas se ejecutan en un navegador incrustado en la aplicación nativa y como se ha explicado anteriormente distintas versiones de navegadores implementan distintas características de HTML5 y eso deriva en inestabilidades en la interfaz gráfica de usuario.

Estas inestabilidades, según se ha comprobado, son más comunes en Android que iOS. Aplicaciones como "3D BMI" las llevan al extremo, inutilizando por completo la aplicación cuando intentamos desplazar una rueda de opciones "estilo iOS" y la interfaz de usuario se ve desplazada fuera de los límites de la pantalla tal y como se aprecia en la siguiente figura.

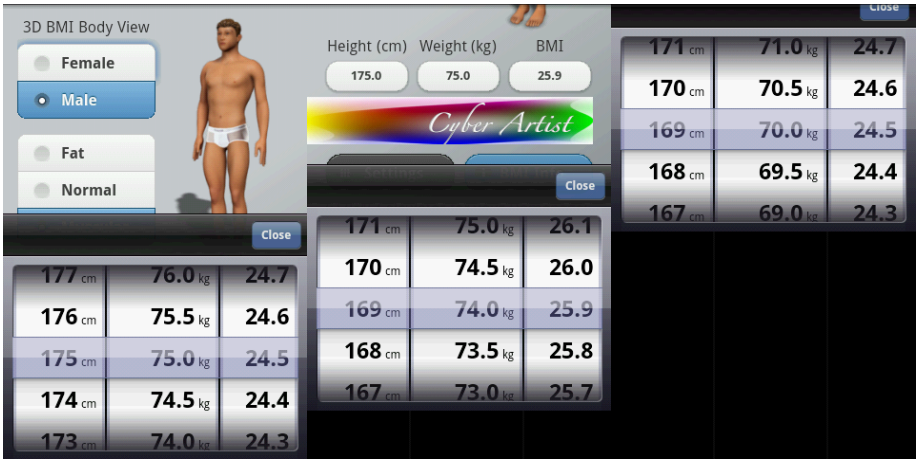


Figura III.3: 3D BMI, comportamiento inestable

Se han llegado a encontrar inestabilidades que impedían la usabilidad de la aplicación desde su inicio, como es el caso de "3-card Brigade", una aplicación de un juego de cartas. Mientras que en iOS su comportamiento es normal, en Android las dimensiones de los límites de la aplicación no se correspondían con los de la pantalla, mostrando únicamente una porción de la imagen aumentada.

Figura III.4: 3-Card Brigade para Android (izq) y para iOS (der)

También cabe debatir si una aplicación debe de tener exactamente el mismo aspecto en todas las plataformas**.** Generalmente sistemas operativos como iOS y Android, ofrecen al desarrollador unas API que permiten acceder a elementos nativos de su interfaz gráfica, como por ejemplo selectores, menús o diálogos de notificación que unifican la apariencia del conjunto de aplicaciones.

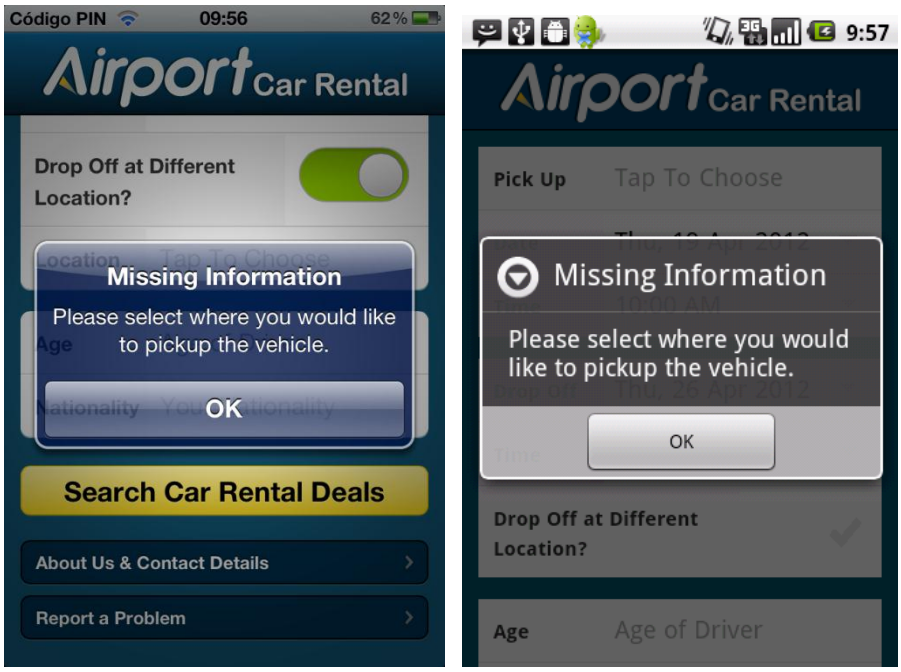


Figura III.5: Airport Car Rental, notificaciones iOS (izq) y Android (der)

Las aplicaciones de PhoneGap analizadas, en su gran mayoría, desarrollan sus propios elementos de interfaz gráfica, que generalmente, están inspirados en iOS, por lo que en un sistema Android destacan por su aspecto fuera de contexto.

Sin embargo, PhoneGap permite al desarrollador acceder a las API nativas del dispositivo. Esto se aprecia en aplicaciones como "Airport Car Rental" donde se usan las notificaciones nativas del sistema

Observamos también como, en el caso de iOS, se usan las notificaciones de la

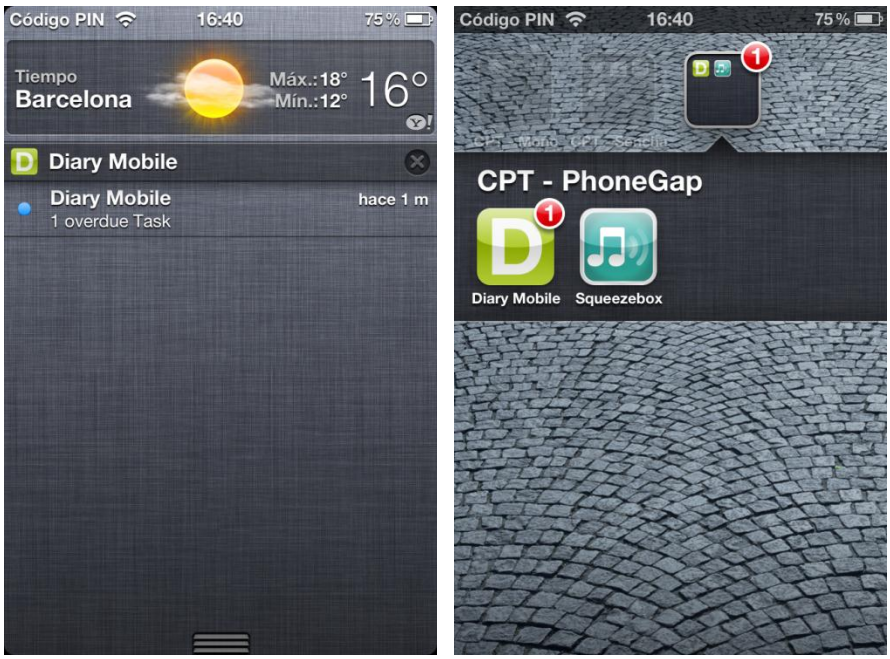


Figura III.6: Notificaciones en el "Centro de notificaciones"

pantalla principal y del "Centro de Notificaciones".

En cambio otras aplicaciones como "Diary Mobile" hacen uso de tecnología web para implementar sus notificaciones.

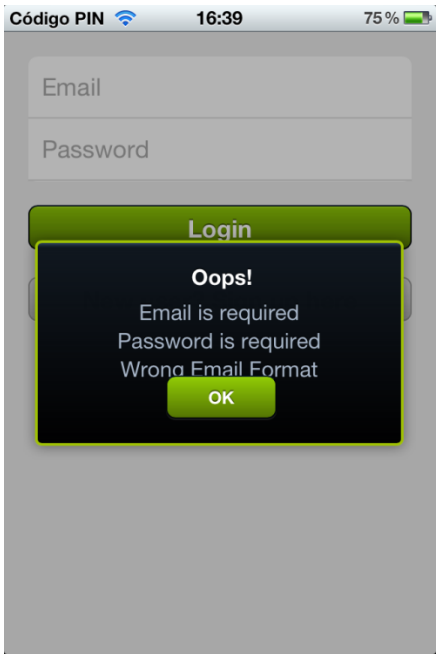


Figura III.7: Diary Mobile,
notificación emergente web

Evaluando el rendimiento de las aplicaciones se observa como es considerablemente más lento que una aplicación completamente nativa. Esto se agrava aún más en el dispositivo Android HTC Magic, donde la ralentización en la respuesta hace que las aplicaciones sean prácticamente inservibles. Ésta ralentización se ve aumentada cuando, con el objetivo de simular el comportamiento de iOS, las aplicaciones incluyen animaciones en sus transiciones de pantalla y componentes de selección, como por ejemplo, ruedas para seleccionar entre un conjunto de valores o interruptores para los campos booleanos.

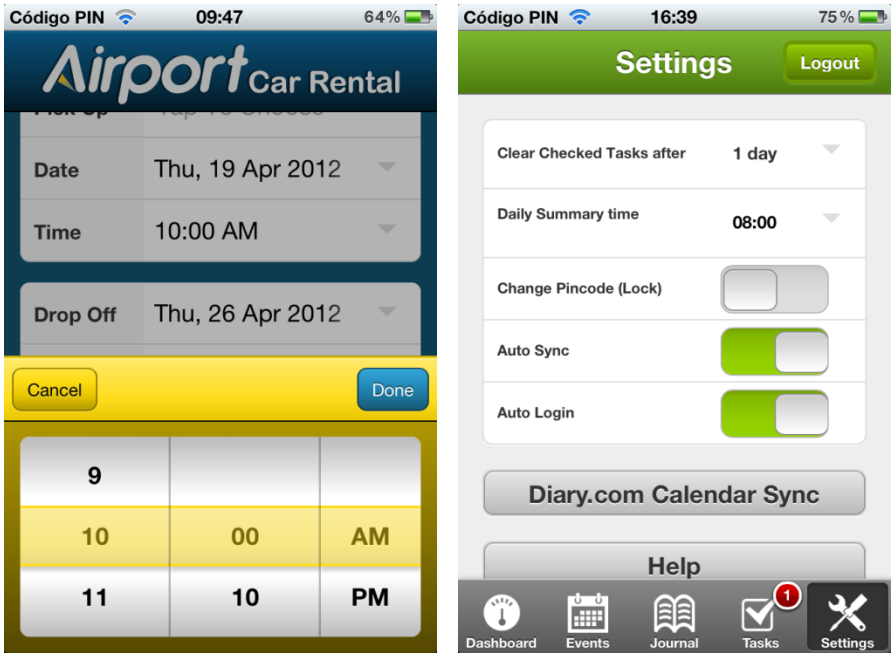


Figura III.7: Elementos de selección de parámetros

2.1.2 Conclusiones

PhoneGap es una plataforma pensada para desarrolladores web que quieren expandirse hacia el mercado de los *Smartphone* sin la necesidad de aprender un lenguaje nuevo o un entorno de desarrollo (IDE) nuevo.

Las aplicaciones se pueden desarrollar enteramente en HTML, JavaScript y CSS, para después incrustarlas en un navegador web sin controles dentro de una aplicación nativa, creando así, una aplicación híbrida.

Sin embargo, la principal aportación del framework PhoneGap es una interfaz Javascript que permite a los desarrolladores acceder a la mayoría de las funcionalidades nativas del dispositivo móvil. Esto es, listas de contactos, aplicaciones de llamada, sensores, GPS, entre otros. A su vez, es gratuito, por lo que reduce los costes, y extensible, por lo que se le pueden añadir funcionalidades nativas y ampliar la interfaz.

Por el contrario, la aplicación también se puede crear, tal y como se ha visto en el apartado anterior en los ejemplos de aplicaciones, sin la necesidad de llamar a esas APIs y usando únicamente elementos web, creando una aplicación más portable.

Asimismo, crear una aplicación totalmente multiplataforma únicamente con elementos web y sin diferencias entre distintas versiones, crea problemas de rendimiento entre dispositivos, ya que los navegadores de iOS y Android no

implementan las mismas especificaciones de HTML5 ni tienen la misma potencia para ejecutar código JavaScript con la debida fluidez.

Tampoco una aplicación para Android debería tener el mismo aspecto que una para iOS, ya que plataformas diferentes usan patrones de estilo diferentes. Ejecutar una aplicación con un estilo completamente iOS en un Android no transmite una buena sensación al usuario.

Para resumir, es una herramienta orientada a desarrollar aplicaciones de manera rápida, siempre que se tenga experiencia en el campo del desarrollo web, para múltiples plataformas (todas las capaces de ejecutar HTML, JavaScript y CSS en su navegador) y accesible ya que la herramienta es gratuita.

2.2 Sencha

Sencha ofrece una variedad de productos para ayudar a los desarrolladores de JavaScript a crear aplicaciones web completas de manera rápida.

Entre sus productos se encuentran **Sencha Touch**, para plataformas móviles (que son las que analizaremos en este apartado), **Ext JS** para navegadores de escritorio y **Ext GWT** para aplicaciones de navegador web de escritorio basadas en **Google Web Toolkit**.

Sencha Touch incluye características que permiten transformar un sitio web en una aplicación web, añadiendo soporte para interacciones táctiles, animaciones y manipulación de datos. Las plataformas soportadas son Android, iOS y Blackberry (dispositivos táctiles).

2.2.1 Análisis

Una gran cantidad de las aplicaciones de Sencha encontradas son del tipo Web Apps, es decir, que se tienen que ejecutar desde un navegador web. Esto implica que no se pueden descargar desde el mercado nativo del dispositivo ni ejecutarse como una aplicación enteramente nativa. Sin embargo, también encontramos aplicaciones de tipo híbridas, es decir, navegador incrustado en una aplicación nativa.

Las aplicaciones presentan las mismas ventajas e inconvenientes que todas las aplicaciones móviles creadas con tecnología web ya comentadas en el apartado de PhoneGap.

Podemos encontrar "*layouts*" inestables, sobretodo en la transición de modo "*portrait*" a "*landscape*". Esto se aprecia en aplicaciones como TrueSkin, donde la aplicación sale de la pantalla, o Xero Touch, donde los botones inferiores desaparecen después de navegar un poco por los menús.

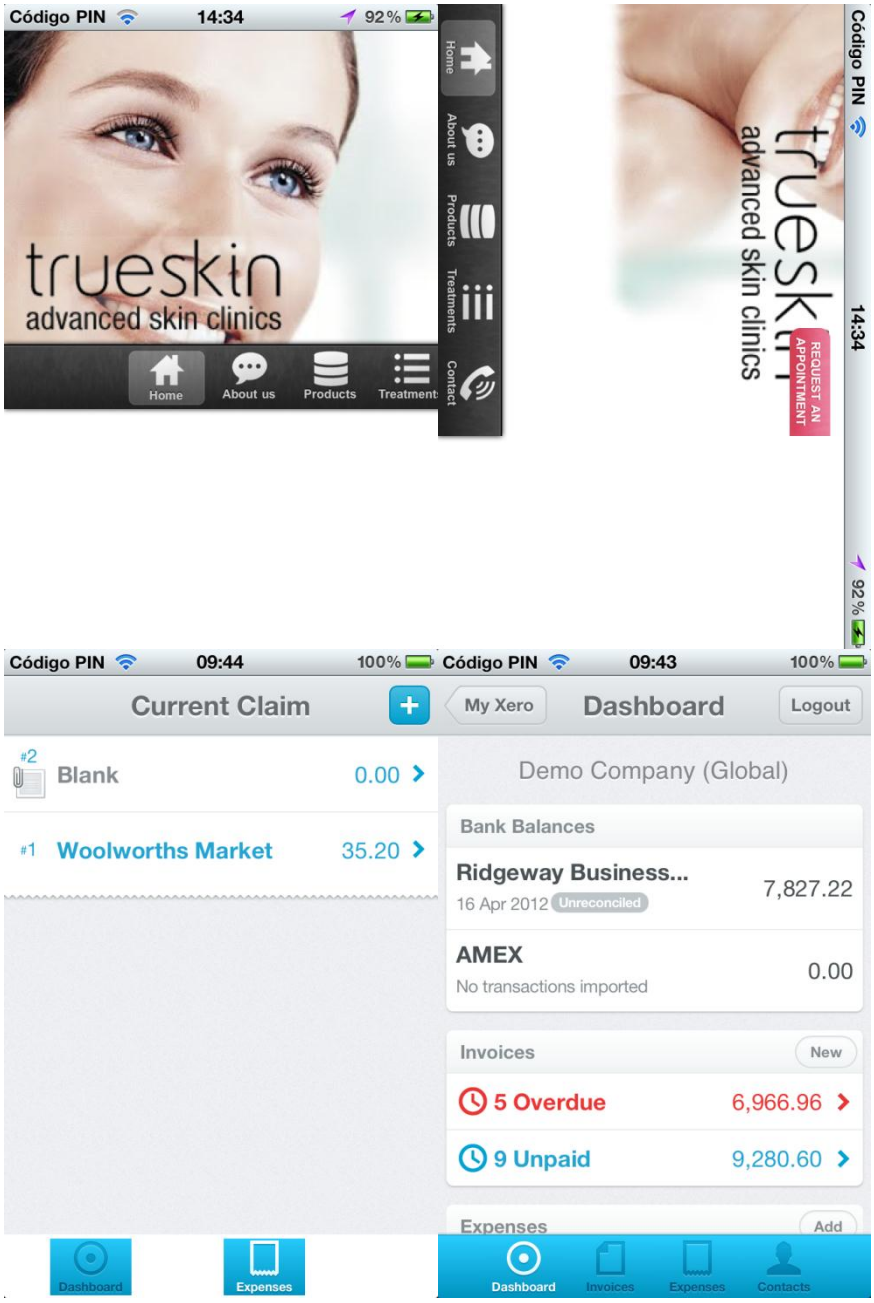


Figura III.8: Layout inestables

El estilo de los selectores de opciones sigue el estilo de iOS, de la misma forma que lo hacen las aplicaciones analizadas en el apartado PhoneGap.

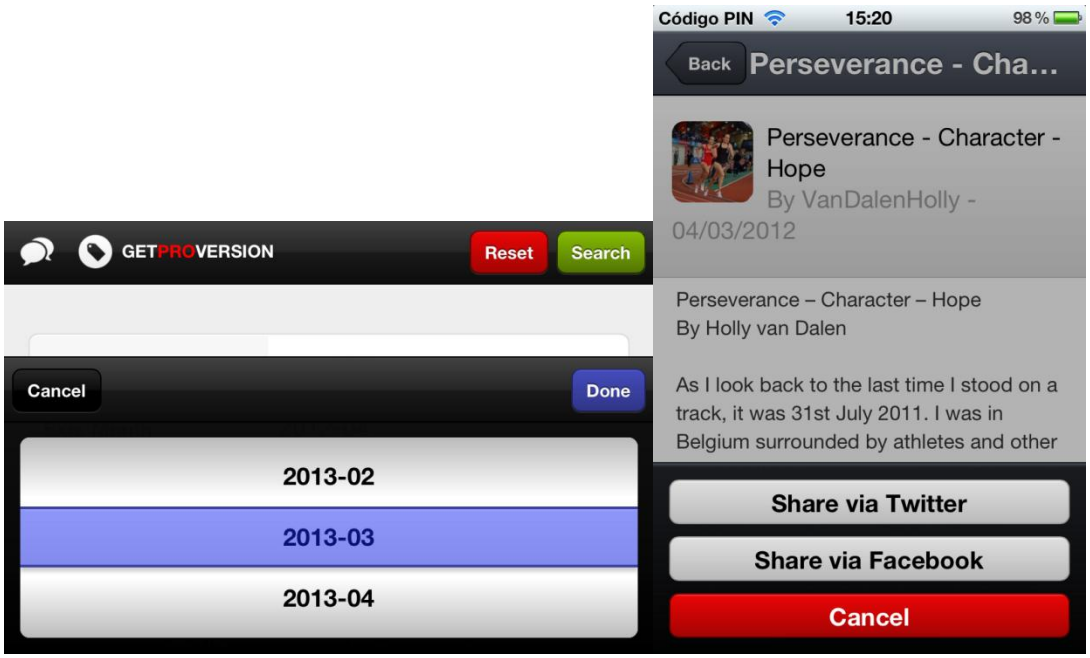


Figura III.9: Componentes de selección estilo iOS

En cuanto al rendimiento, al ser código web el que realmente se ejecuta, depende de las capacidades del *Smartphone*, de su navegador y de su fluidez.

Asimismo, Sencha ofrece acceso a las APIs del teléfono, aunque de una forma insuficiente, según sus desarrolladores. Se puede, por ejemplo usar el sistema de notificaciones nativas como en PhoneGap o bien usar un sistema de notificaciones propio de Sencha si no se va a crear una aplicación híbrida.

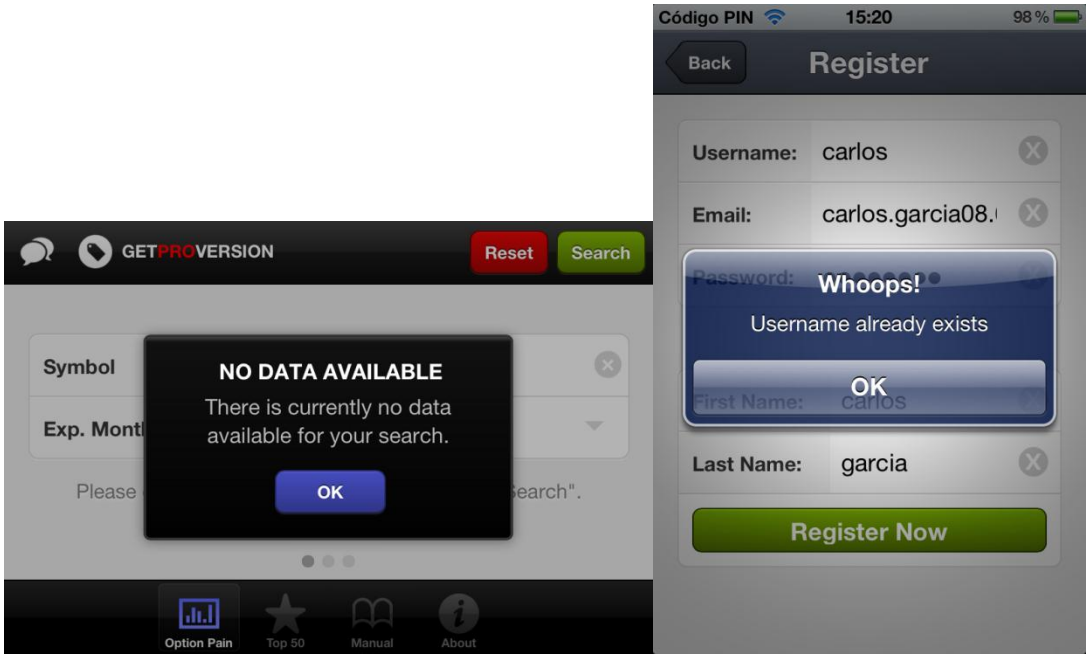


Figura III.10: Notificaciones nativas y de HTML5

2.2.2 Conclusiones

Sencha Touch, al ser una plataforma para crear aplicaciones web móviles mediante HTML, JavaScript y CSS, presenta las mismas ventajas e inconvenientes que su competidor, PhoneGap.

Al contrario que PhoneGap, no es gratuita aunque dispone de un IDE propio (Sencha Architect), por lo que se facilita la creación de la interfaz de usuario, permitiendo crear aplicaciones vistosas fácilmente (siguiendo el patrón de estilo de iOS).

2.3 Xamarin (MonoTouch y Mono for Android)

Las herramientas de **Xamarin** permiten a los desarrolladores de **Microsoft .NET** especializados en C# y librerías de .NET crear aplicaciones para las plataformas iOS (mediante MonoTouch) y Android (mediante Mono for Android).

Aunque Xamarin no ofrezca soporte directo para plataformas Windows Phone, todo el código de .NET y C# puede ser ejecutado en Windows Phone 7.

Al contrario que las herramientas previamente analizadas, Xamarin no crea aplicaciones web ni aplicaciones híbridas, sino que crea aplicaciones que se ejecutan de forma completamente nativa mediante un **runtime**.

Las aplicaciones creadas con **MonoTouch** son pre-compiladas en ejecutables nativos de iOS. **MonoDroid** para Android usa compilación Just in Time (JIT) desplegando un entorno de ejecución incrustado en la aplicación del sistema.

Los desarrolladores tienen disponible un IDE llamado **MonoDevelop** para crear sus aplicaciones. Aunque, en caso de estar desarrollando para Android en Windows, también se puede usar Microsoft Visual Studio.

Ambos productos, MonoTouch y Mono for Android, están disponibles en versión de prueba por tiempo ilimitado. La única restricción es que las aplicaciones sólo se pueden desplegar en emulador. Se pueden obtener licencias anuales de la versión profesional a partir de 400 US$.

2.3.1 Análisis

Las aplicaciones desarrolladas con Xamarin son en su gran mayoría, aplicaciones para el sistema de Apple, debido a que se orienta a desarrolladores de C# que quieren desarrollar aplicaciones nativas para iOS sin la necesidad de aprender Objective-C. Inicialmente únicamente estaba disponible el producto MonoTouch para iOS que fue lanzado en 2009. Mono for Android no se lanzaría hasta 2011 permitiendo también alcanzar a los usuarios del sistema de Google.

Al no usar código web para crear las aplicaciones, únicamente se puede reaprovechar la parte no correspondiente a la interfaz gráfica. Esto implica que no se puede exportar un mismo proyecto a ambos dispositivos, como ocurría con las herramientas anteriormente analizadas. Por lo que el resultado es un producto orientado a desarrolladores de software avanzados en lugar de a desarrolladores web.

El diseño de la interfaz de cada aplicación necesita ser creado en su propio entorno. Por ejemplo, para una aplicación de iOS necesitaremos tener un Mac

con XCode instalado y su editor de interfaces gráficas. El código generado se exportará a MonoDevelop en C# donde se permitirá al desarrollador seguir implementando la lógica de la aplicación.

Estas aplicaciones no muestran ninguna pérdida de rendimiento, como pasaba con las híbridas analizadas anteriormente, ya que se ejecutan de una manera absolutamente nativa, y no mediante navegadores que ejecutan JavaScript, por lo que el usuario observará un comportamiento mucho más fluido.

En las pruebas realizadas en HTC Magic las aplicaciones corrían perfectamente, mientras que cuando se hacían las pruebas con PhoneGap y Sencha la gran mayoría eran inestables e incluso demasiado lentas para ser usadas.

En cuanto al diseño, como se ha comentado antes, cada aplicación muestra su propio estilo (acorde con la plataforma) aunque internamente puedan estar reaprovechando código. En las siguientes capturas podemos apreciar las mismas pantallas de distintas versiones de aplicaciones para iOS (izquierda) y Android (derecha).

En el siguiente par de imágenes vemos como las notificaciones emergentes de la aplicación Logbook Pro son completamente nativas.

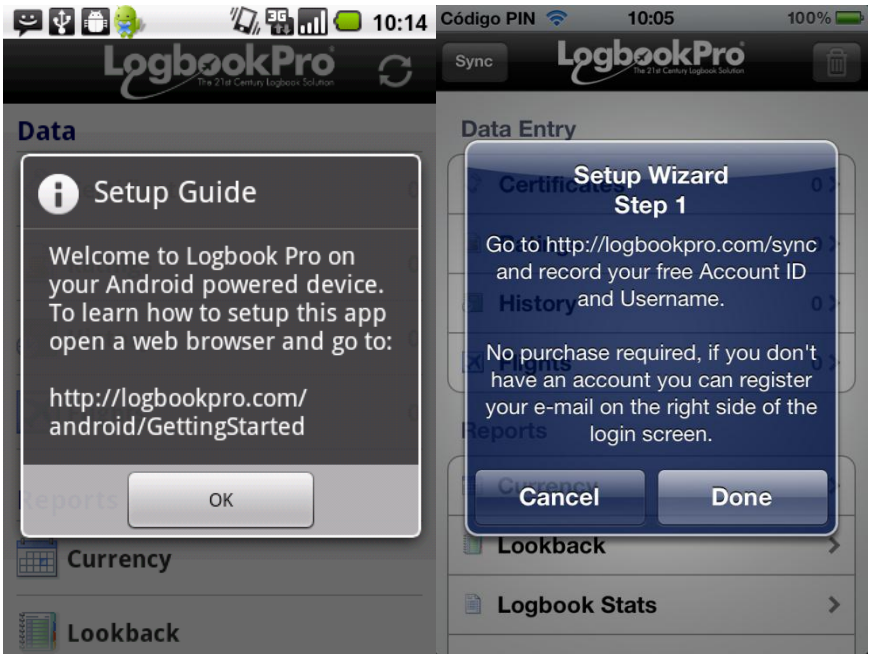


Figura III.11: Notificaciones nativas, Android (izq) y iOS (der)

En cuanto a las diferentes pantallas vemos que cada plataforma adapta su interfaz.

Figura III.12: Interfaces nativas, iOS (izq) y Android (der)

Otra aplicación ejemplo puede ser USTA.TV, donde vemos que se ofrecen las mismas funcionalidades con diferentes presentaciones.

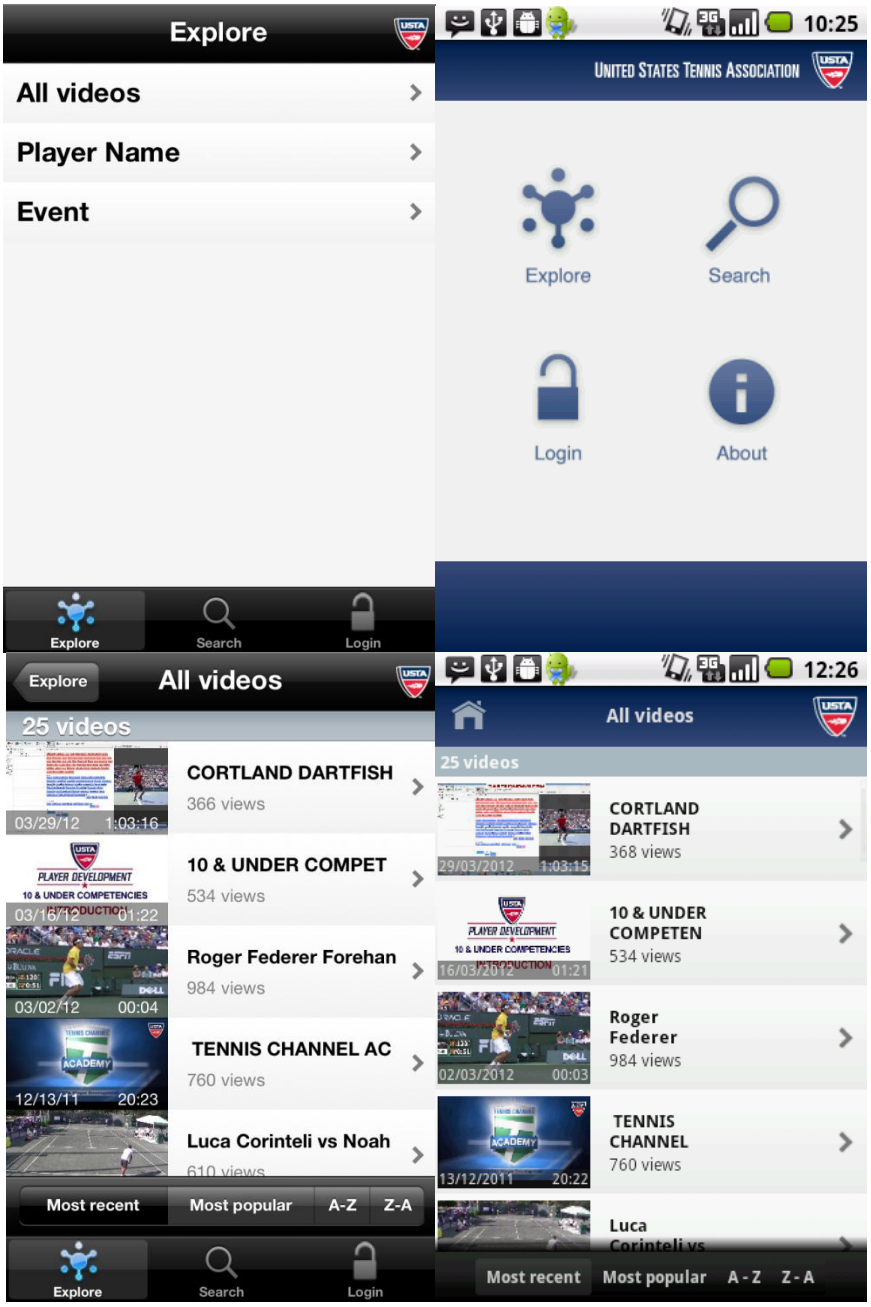


Figura III.13: Interfaz aplicación USTA.TV, iOS (izq) y Android (der)

2.3.2 Conclusiones

Tal y como dicen los creadores de Xamarin:

"Xamarin's cross-platform mobile app development software allows .NET/C# programmer Access to all native APIs with their existing skills and tooling to cut 70% of app development time and costs by sharing up to 90% of high-performance mobile app code running on iOS, Android and Windows Phones, Tablets and embedded devices"

Sin estar orientado directamente a la creación de aplicaciones multiplataforma, sí que se permite una alta reutilización del código no perteneciente a la interfaz gráfica. Esto agiliza el proceso tanto económicamente como temporalmente, sin las limitaciones de las aplicaciones web híbridas.

Si se posee un alto conocimiento del lenguaje y librerías de Microsoft, crear aplicaciones para iOS (incluso sin el ánimo de alcanzar otras plataformas), es un proceso más rápido que usando Objective-C y XCode. Según dicen los desarrolladores para Xamarin, las APIs ofrecidas por la herramienta simplifican muchas acciones que llevarían más líneas de código con el lenguaje de Apple.

Sin embargo, al estar enfocado a desarrolladores de software avanzados, el proceso es más complicado que usando las herramientas de desarrollo web comentadas en los anteriores apartados, aunque ellas sacrifiquen drásticamente el rendimiento de la aplicación.

Resumiendo, los productos de Xamarin, MonoTouch y Mono for Android, permiten a los desarrolladores crear aplicaciones para Android, iOS y WP7 (implícitamente, ya que para WP7 se desarrolla en .NET) utilizando un lenguaje moderno y avanzado como es C#/.NET. Fomentando el reutilizamiento de código entre versiones sin tener que empezar de cero en cada dispositivo.

A pesar de ello, será necesario tener conocimientos del funcionamiento de las APIs de iOS y Android ya que la API de Xamarin en C# replica muchas de las llamadas nativas.

2.4 Consideraciones

Las Cross-Platform Tools son una gran herramienta que simplifica la vida a muchos desarrolladores de software. Permiten crear aplicaciones de manera más rápida para múltiples plataformas sin tener que invertir tiempo en aprender un lenguaje de programación específico ni familiarizarse con un entorno de desarrollo determinado.

Son de gran utilidad tanto para empresas, como para desarrolladores aficionados que quieren alcanzar el máximo número de usuarios posible con sus productos o creaciones invirtiendo el menor dinero y tiempo posible.

Sin embargo hay que considerar que en la mayoría de los casos la calidad de la aplicación se ve comprometida debido a un falta de acceso a las API nativas del dispositivo, que permitirían al desarrollador aprovechar recursos que de otra manera son inaccesibles, o debido a una pobre interfaz gráfica que no se adapta a los patrones de diseño establecidos por el fabricante del dispositivo y especificados en sus API y manuales de estilo.

Las críticas a las Cross-Platform Tools habitualmente van orientadas a la dependencia de un tercer componente entre el desarrollador y el fabricante del dispositivo. Esta dependencia hace que las aplicaciones creadas con el *framework* de terceros estén siempre "un paso por detrás" debido al retraso del CPT en actualizar sus API adaptándolas a las nuevas versiones.

Para los CPT que permiten desplegar las aplicaciones de manera automática a distintas plataformas este problema se agrava cuando su potencia es el mínimo común denominador de todas las plataformas que soporta.

Estas críticas quedan fuertemente mencionadas en la carta abierta de Steve Jobs (ex CEO de Apple) "*Thoughts on Flash*" donde se muestra el motivo por el cual el *plugin* **Adobe Flash** no está disponible para los dispositivos iOS.

Flash también es un CPT que, aunque ahora está siendo abandonado en favor de HTML5, permitía mediante su *plugin* ejecutar aplicaciones en cualquier plataforma que lo soportara, siendo completamente propietario y comprometiendo la calidad de sus aplicaciones a una tercera empresa independientemente del dispositivo físico donde se ejecutaran.

Las Cross-Platform Tools son herramientas muy potentes e indiscutiblemente van a acabar siendo el futuro de un mercado donde cada vez se busca alcanzar el mayor público posible. Esto es posible únicamente, con un coste económico y temporal aceptable, mediante herramientas de desarrollo multiplataforma.

Sin embargo, siempre que se requieran aplicaciones como juegos, aplicaciones multimedia avanzadas que requieran una intensa interacción con el usuario o aplicaciones que usen características específicas de una plataforma, los desarrolladores no van a tener otra opción que usar los SDK nativos para sacar el mayor provecho y rendimiento.

3 Conclusión

Después de estudiar y analizar las CPT más destacadas del mercado, creando aplicaciones de prueba en distintas plataformas y estudiando el flujo de desarrollo de cada una, se decidió escoger el *framework* **PhoneGap**.

La decisión se justifica dado que el propósito del proyecto es el de crear un *framework* con funcionalidades de seguridad y PhoneGap es el más accesible y extensible, ya que permite crear *plugins* nativos donde se desarrollarían las funcionalidades del *framework*.

Siendo cierto que las aplicaciones web híbridas que se crean con PhoneGap presentan algunos problemas de rendimiento, también es cierto que es el *framework* más usado por los desarrolladores, dada la facilidad con la que permite crear aplicaciones. Esto permitiría ofrecer a la comunidad desarrolladora funcionalidades de seguridad, que normalmente su complejidad representa un inconveniente, como al firma electrónica de forma sencilla y accesible.

IV - DISEÑO DEL PROYECTO

1 Introducción

Este capítulo se centra en la etapa de diseño de los diferentes componentes del proyecto y su arquitectura general.

2 Requerimientos y componentes

Desarrollar un *framework* de seguridad y verificar su funcionamiento mediante un demostrador requiere del diseño e implementación los siguientes componentes.

- Una aplicación móvil que actué de demostrador del *framework*.
- Un servidor que complemente a la aplicación móvil ofreciendo servicios de autenticación, almacenamiento y asistencia en los procedimientos de firma.
- Una base de datos para almacenar los usuarios y sus sesiones en los diferentes servicios.

El *framework* está orientado a dispositivos móviles por lo que el demostrador tiene que ser una aplicación para teléfonos inteligentes y/o tabletas.

2.1 Diseño de la aplicación móvil

La aplicación actúa de demostrador del *framework* de firma por lo que su principal funcionalidad es la de firmar los documentos PDF que seleccione el usuario. Estos documentos se podrán firmar con claves almacenadas en el dispositivo, utilizando internamente el *framework* implementado, o mediante el servicio de firma de *TrustedX*.

2.1.1 Interfaz gráfica

Se ha diseñado la interfaz gráfica para que sea lo más sencilla e intuitiva posible. Minimizando el número de pantallas y utilizando elementos y componentes de fácil interacción por parte del usuario.

La primera pantalla que se encuentra el usuario la primera vez que inicia la aplicación es la de bienvenida.

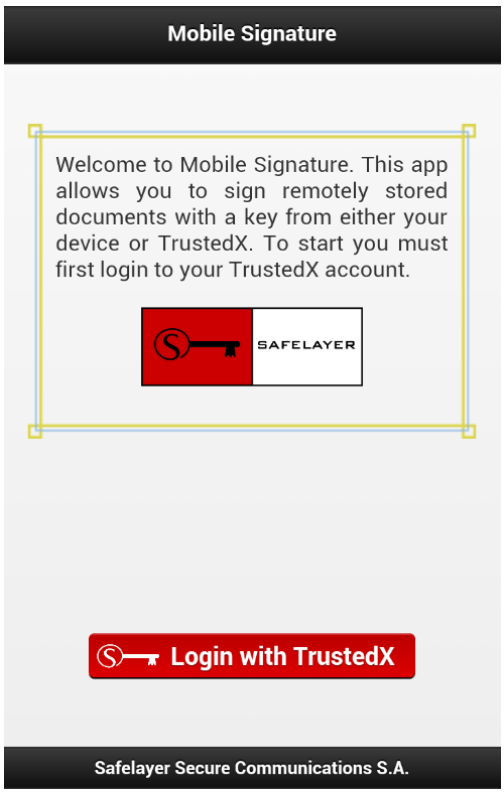


Figura IV.1 Pantalla de bienvenida

Tal y como se aprecia en la figura, se compone de un texto explicativo y un botón para iniciar el proceso de autenticación mediante *TrustedX*.

Una vez el usuario está correctamente autenticado se le mostrará la pantalla inicial.

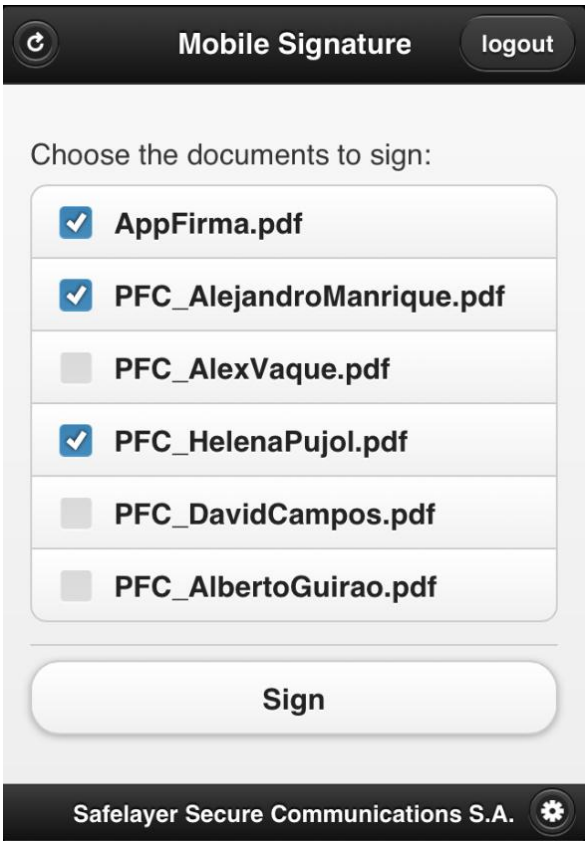


Figura IV.2 Pantalla inicial

En esta pantalla se muestran los documentos del usuario dentro de un contenedor de selección múltiple. Desde esta posición se pueden realizar cuatro acciones diferentes:

- *Refrescar los documentos*, mediante el botón **superior izquierdo**.
- *Cerrar la sesión*, mediante el botón **superior derecho** con el texto **logout**.
- *Firmar los documentos seleccionados*, mediante el botón **inferior** con el texto **Sign**.
- *Acceder a las opciones de configuración*, mediante el botón **inferior derecho**.

La siguiente pantalla que el usuario tiene disponible es la de configuración.

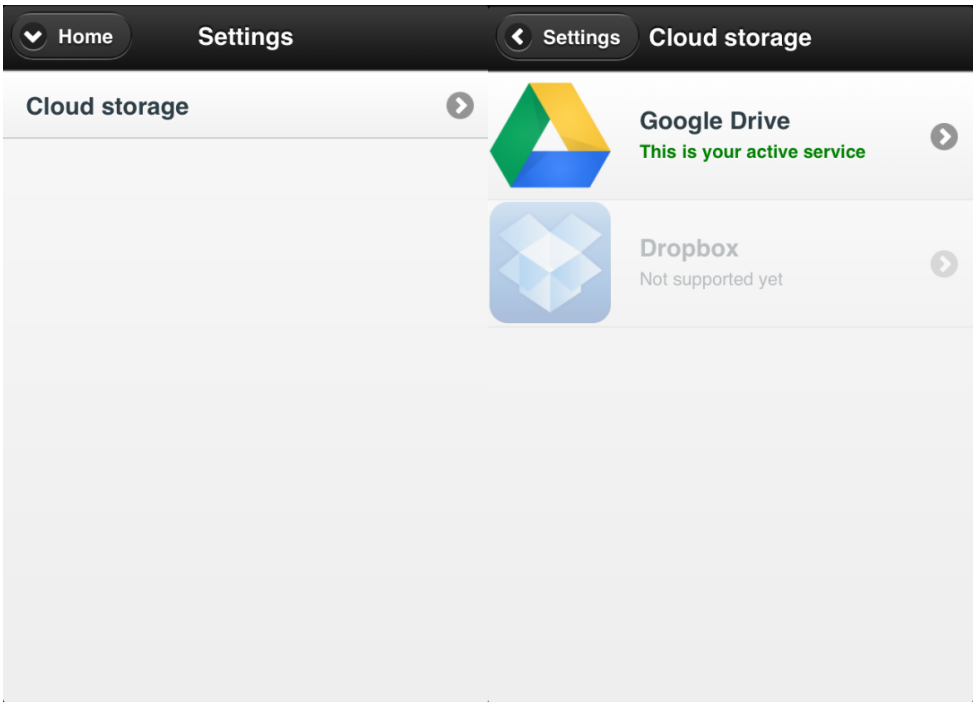


Figura IV.3: Pantallas de opciones

Aquí el usuario puede configurar los servicios de almacenamiento en la nube que le permitirán tener acceso a los documentos a firmar. La siguiente pantalla permite que el usuario se autentique en el servicio Google Drive.

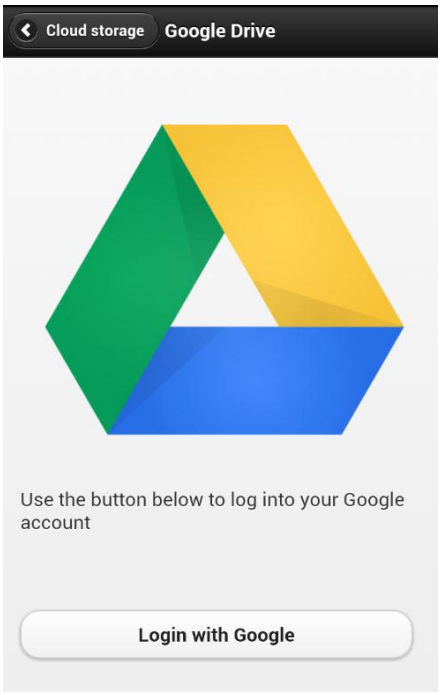


Figura IV.4: Pantalla Google Drive

2.1.2 Diagramas de flujo de la aplicación

En esta sección se muestran los posibles comportamientos de la aplicación móvil mediante diagramas de flujo.

2.1.2.1 Flujo inicial

El flujo básico que transcurre cuando el usuario inicia la aplicación por primera vez.

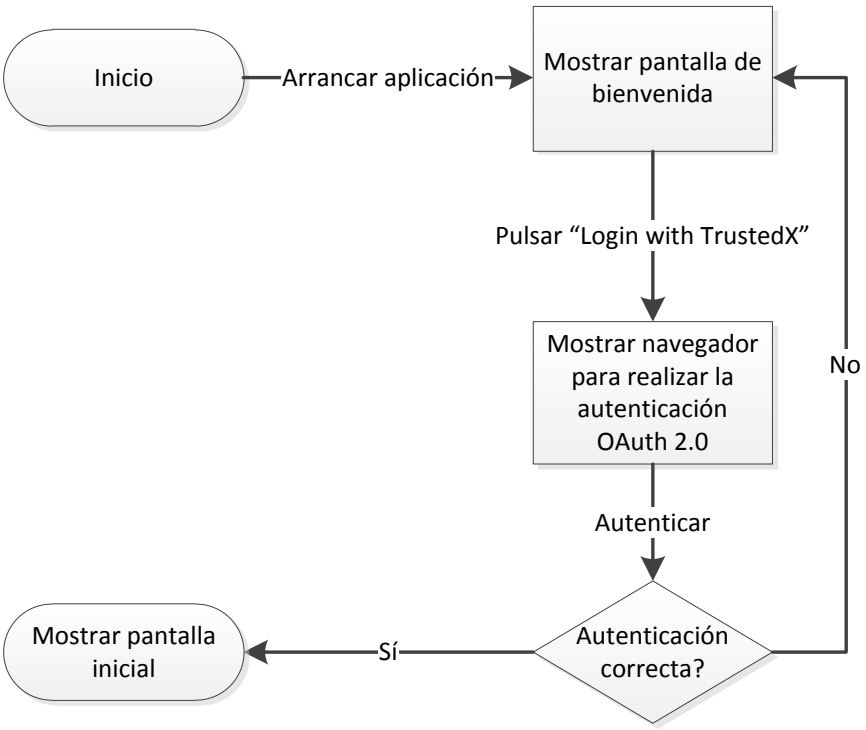


Figura IV.5: Flujo inicial

2.1.2.2 Autenticación con el servicio de almacenamiento

Antes de que el usuario pueda realizar ninguna acción con los documentos, primero tiene que dar de alta un servicio de almacenamiento en la nube como por ejemplo Google Drive.

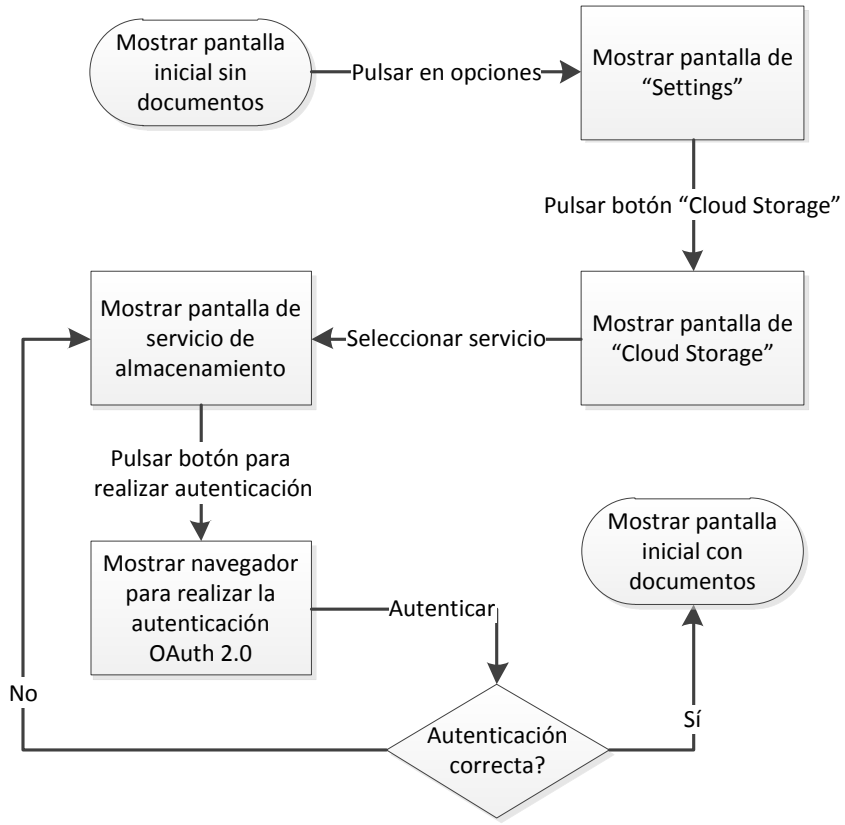


Figura IV.6: Autenticación con el servicio de almacenamiento

2.1.2.3 Firmar documentos

Una vez el usuario está autenticado en TrustedX y en el servicio de almacenamiento ya puede firmar los documentos que se muestran.

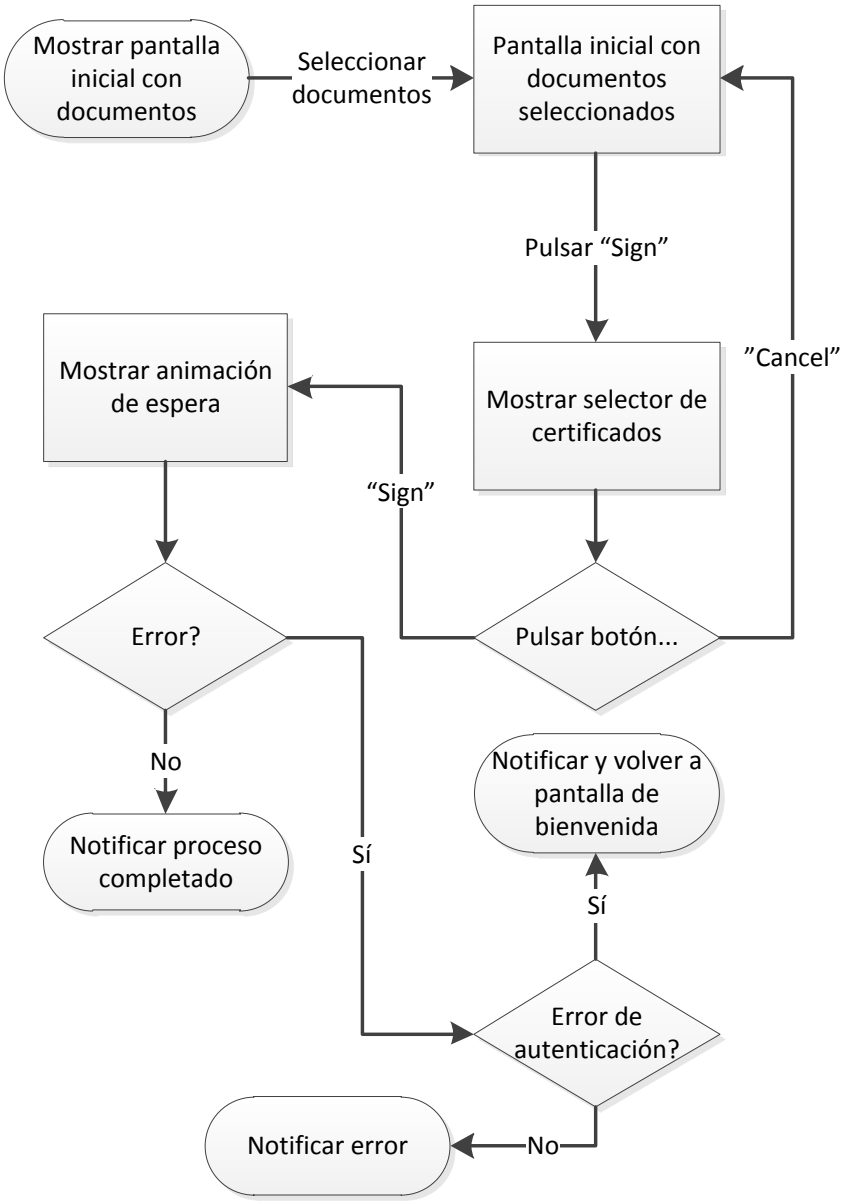


Figura IV.7: Firmar documentos

2.1.3 Casos de uso

El siguiente diagrama muestra los casos de uso del demostrador.

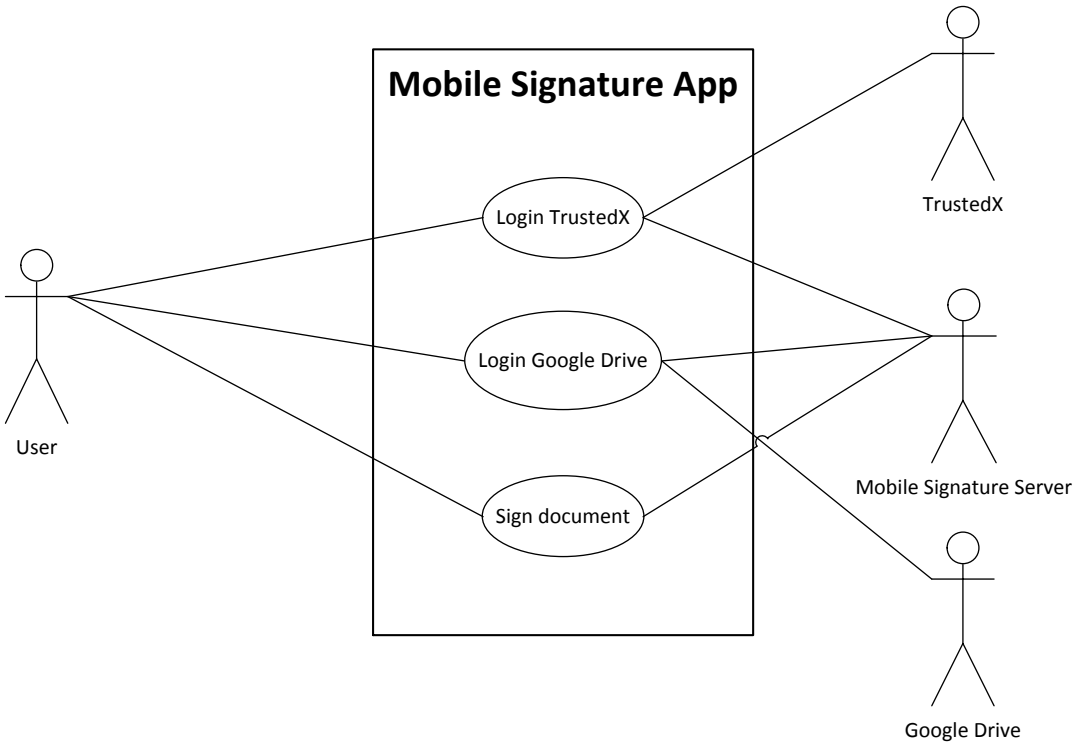


Figura IV.8: Casos de uso de la aplicación móvil

Los actores que aparecen son el usuario final de la aplicación móvil (User), el servidor asociado a la aplicación móvil, que también forma parte del desarrollo de este proyecto y los dos servicios externos, TrustedX y Google Drive (escalable a otros servicios de almacenamiento).

El servidor interviene en los casos de uso de autenticación con los servicios externos porque al usar el flujo *code grant* de OAuth 2.0 es el que actúa de *Authorization Server* y recibe el *token* final.

2.1.3.1 Descripción de los casos de uso

Las siguientes tablas contienen las descripciones de los casos de uso del diagrama.

Caso de uso	Login TrustedX
Actores	Usuario, TrustedX, Mobile Signature Server (MSS)
Descripción	Este caso de uso describe como el usuario inicia sesión en la aplicación a través de **TrustedX**.
Precondición	Ninguna
Postcondición	Si el caso de uso finaliza satisfactoriamente, el **usuario** inicia sesión en el sistema. Si no, el sistema no varía.
Flujo básico	1. El **usuario** pulsa sobre el botón *Login with TrustedX*. 2. La aplicación abre un navegador web que obtiene un formulario de autenticación de **TrustedX**. 3. El **usuario** escribe sus datos de autenticación y los envía. 4. **TrustedX** valida los datos de autenticación del **usuario** y lo autentica. 5. **MSS** recibe el token de autenticación y redirige el navegador a una web de autenticación satisfactória. 6. La aplicación detecta la autenticación satisfactoria, cierra el navegador web y le concede acceso al usuario.
Flujos alternativos	**Datos de autenticación inválidos:** Si los datos de autenticación introducidos en el flujo básico no son correctos, la autenticación no se producirá y el sistema no cerrará el navegador. El **usuario** puede decidir volverlo a intentar cerrando el navegador y volviendo al paso 1 del flujo básico.

Caso de uso	Login Google Drive
Actores	Usuario, Google Drive, Mobile Signature Server (MSS)
Descripción	Este caso de uso describe como el usuario inicia sesión en **GoogleDrive**
Precondición	El usuario está autenticado en la aplicación vía TrustedX
Postcondición	Si el caso de uso finaliza satisfactoriamente el usuario inicia sesión en Google Drive. Si no, el sistema no varía.
Flujo básico	1. El **usuario** inicia la autenticación en **GoogleDrive**. 2. La aplicación abre un navegador web con un formulario de autenticación de **Google Drive**. 3. El **usuario** escribe sus datos de autenticación y los envía. 4. **Google Drive** muestra un mensaje advirtiendo de los permisos que el usuario está a punto de dar. 5. El usuario acepta los permisos. 6. **MSS** recibe el *token* de autenticación y redirige el navegador web a una web de autenticación satisfactória. 7. La aplicación detecta la redirección y cierra el navegador web. 8. El caso de uso finaliza satisfactoriamente mostrando la pantalla inicial con los documentos almacenados en Google Drive (si los hay).
Flujos alternativos	**Usuario no acepta los permisos:** Si el usuario no concede permisos a **Google Drive** la autenticación no se produce y el navegador no se cierra. El usuario puede volver a intentar la autenticación cerrando el navegador. **Datos de autenticación inválidos** Si los datos de autenticación introducidos en el flujo básico no son correctos, la autenticación no se producirá y el sistema no cerrará el navegador. El **usuario** puede decidir volverlo a intentar cerrando el navegador y volviendo al paso 1 del flujo básico.

Caso de uso	Sign document
Actores	Usuario, TrustedX, Mobile Signature Server (MSS), *Google Drive*.
Descripción	Este caso de uso describe como la aplicación firma los documentos del usuario.
Precondición	El **usuario** tiene que tener documentos y certificados en el dispositivo y/o en **TrustedX**.
Postcondición	Los documentos seleccionados son firmados con la clave especificada por el **usuario**.
Flujo básico	1. El **usuario** selecciona uno o más documentos de la lista y pulsa sobre el botón *Sign*. 2. La aplicación muestra los certificados con los que se puede firmar los documentos. 3. El **usuario** pulsa sobre un certificado e inicia el proceso de firma. 4. **MSS** se descarga el documento de **Google Drive** y calcula su *hash*. 5. La firma es realizada por el dispositivo o por **TrustedX**, si el certificado seleccionado le pertenece. 6. **MSS** compone la firma en el documento y lo almacena en **Google Drive**. 7. El caso de uso finaliza satisfactoriamente cuando la aplicación le muestra al **usuario** un mensaje de que las firmas se han realizado correctamente.
Flujos alternativos	**Cancelar firma** Mientras se muestran los certificados, el **usuario** puede cancelar y volver al estado inicial sin que se realice ninguna firma. **Usuario no autenticado** La sesión del **usuario** puede haber caducado durante el proceso, en cuyo caso una vez iniciada la firma, se mostrará un mensaje al **usuario** diciéndole que su sesión ha caducado y se le redirigirá a la pantalla de bienvenida.

2.2 Diseño del servidor

La aplicación móvil demostradora del *framework* de firma, requiere de un servidor dedicado con las siguientes funcionalidades:

- Calcular los valores de los *digest* de los documentos a firmar y mandarlos al servicio de firma cuando se requiera.
- Cuando reciba las firmas de los documentos, montar los PDF firmados con dichas firmas.
- Gestionar el proceso de autenticación OAuth 2.0 con *TrustedX* actuando como cliente del flujo de concesión *code grant*.
- Gestionar el proceso de autenticación OAuth 2.0 con Google y otros servicios de almacenamiento, actuando como cliente del flujo de concesión *code grant*.
- Gestionar los diferentes usuarios que utilicen la aplicación mediante una base de datos, así como los *tokens* OAuth 2.0 de sus respectivos servicios, inhabilitándolos en caso de expiración.
- Gestionar los documentos de los usuarios almacenados en un servicio en la nube.

2.2.1 Diseño de la interfaz de autenticación OAuth 2.0 con *TrustedX*

Para implementar la funcionalidad de autenticación con *TrustedX* ha sido necesario crear una interfaz web con un formulario adaptado a sistemas móviles.

El diseño de la interfaz de autenticación para móviles se basa en la versión para navegadores de escritorio. Consiste en un formulario de usuario y contraseña básico y uno para introducir una contraseña de un solo uso, ya que se usa autenticación en 2 pasos.

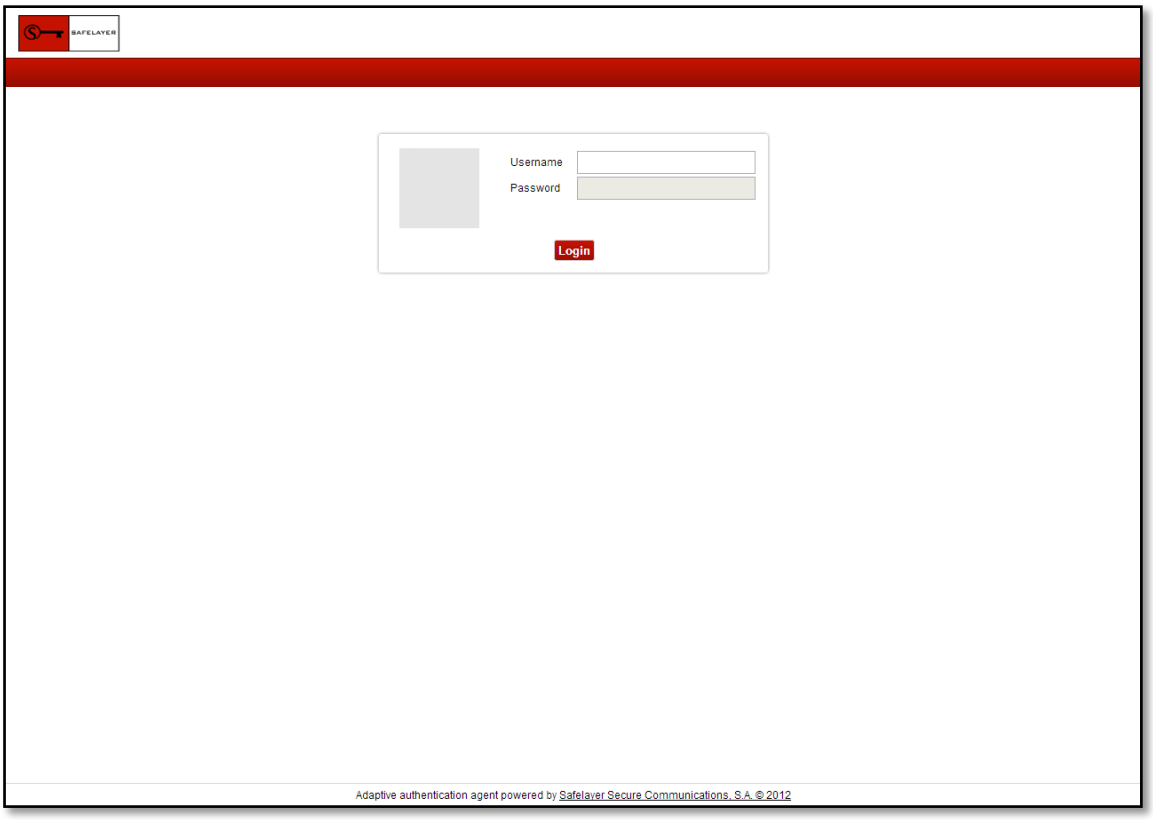


Figura IV.9: Formulario de autenticación en TrustedX, versión escritorio

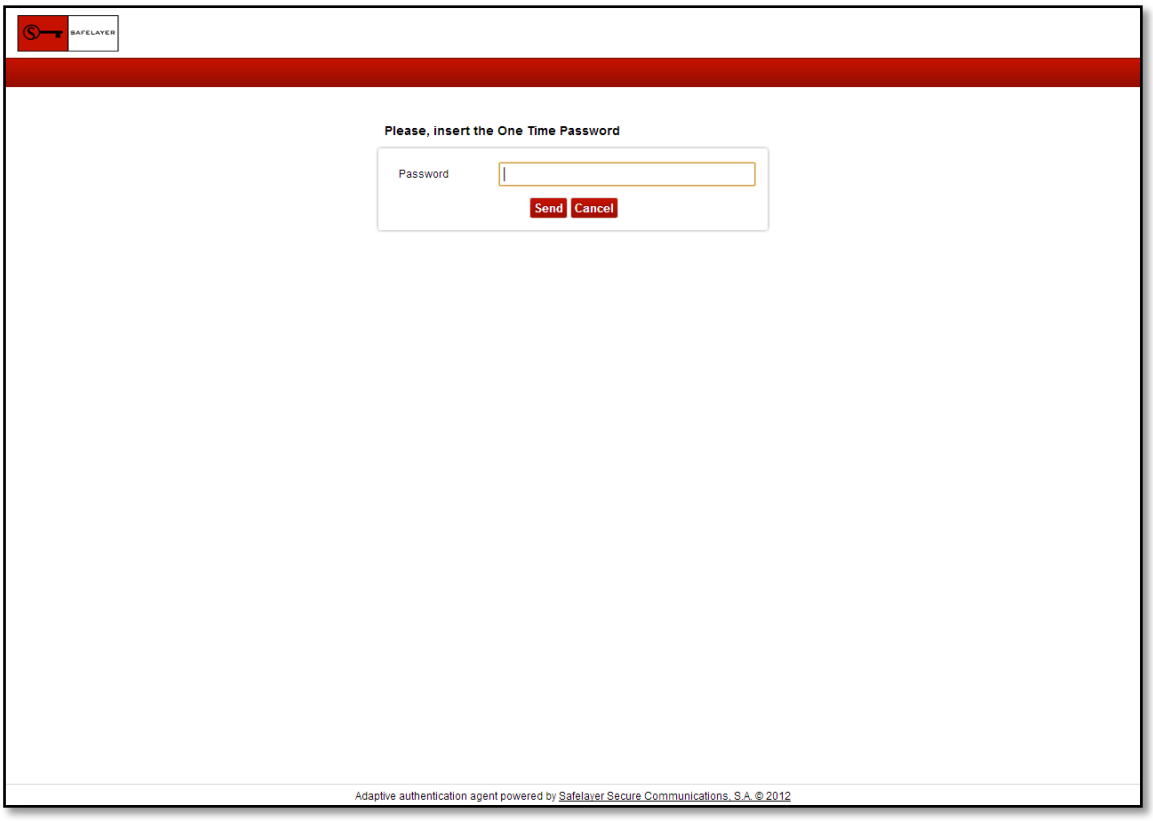


Figura IV.10: Formulario de introducción de OTP, versión escritorio

El dispositivo, en lugar de bajar el formulario del servidor de *TrustedX*, se lo bajará del servidor del proyecto, ya que el formulario de autenticación para móviles no está incorporado de manera oficial en el producto. La siguiente figura muestra los formularios adaptados para pantallas móviles.

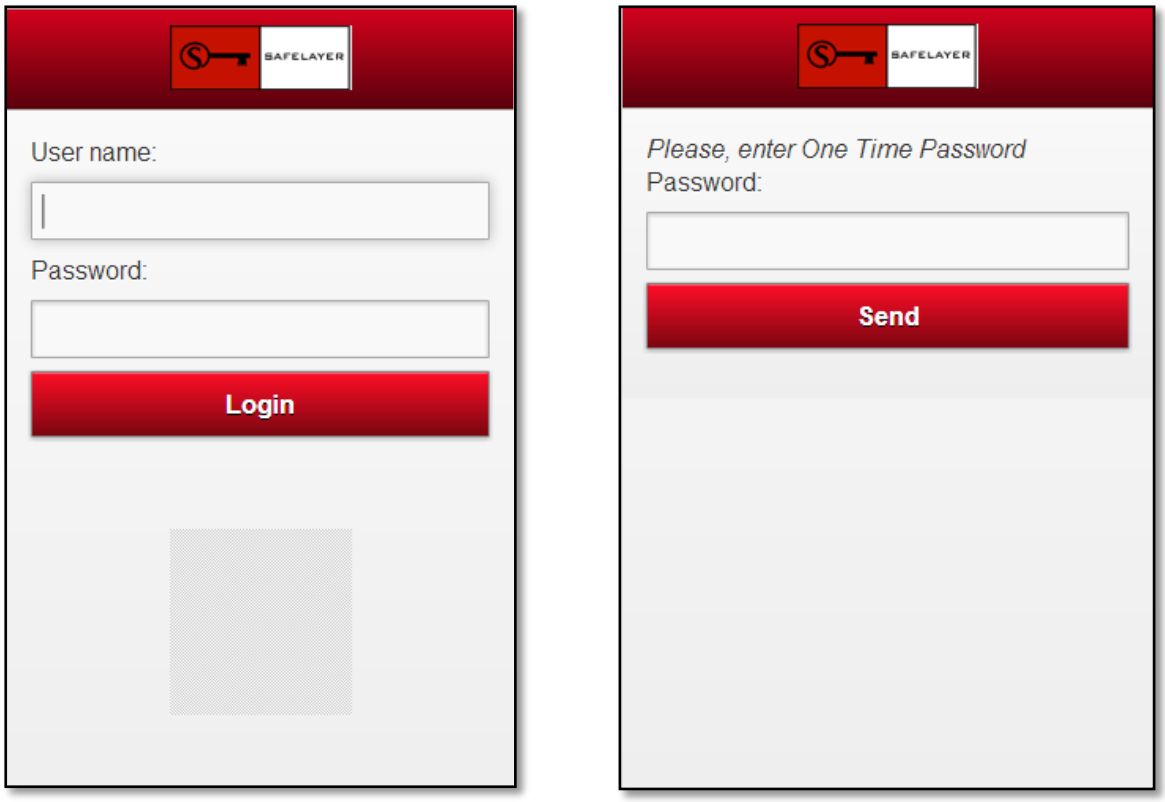


Figura IV.11: Versión móvil de los formularios de autenticación en TrustedX

2.2.2 Casos de uso

La siguiente figura muestra el diagrama de casos de uso que se ha diseñado para el servidor.

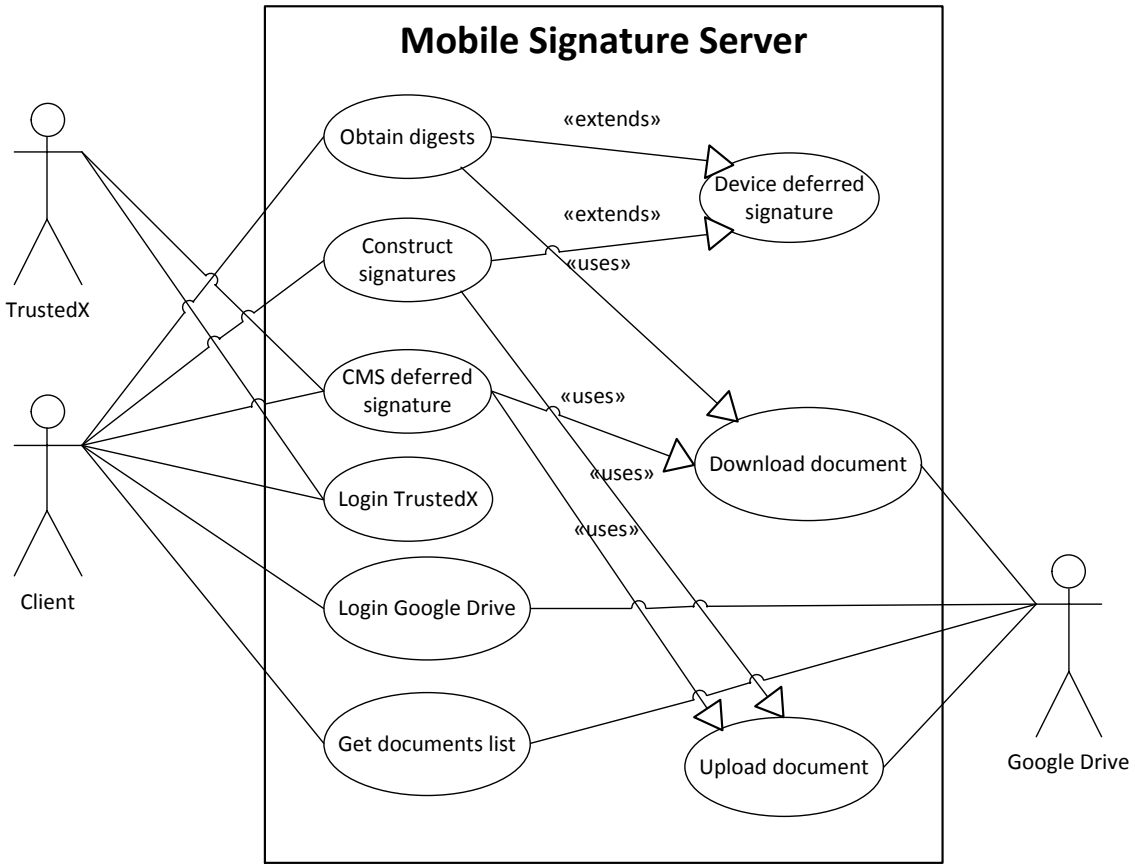


Figura IV.12: Casos de uso del servidor

El sistema tiene tres actores principales que interactúan con él:

- **El cliente** (*client* en el diagrama): En el contexto del proyecto es la aplicación móvil demostradora del *framework* de firma que se ha especificado en la anterior sección.

- **TrustedX**: El servicio de firma, autenticación cifrado de Safelayer Secure Communications. Actúa como IdP y como servicio de firma para el caso de uso *CMS deferred signature*.

- **Google Drive**: El servicio de almacenamiento usado en el contexto del proyecto. Se encarga de almacenar los documentos sin firmar y firmados del cliente o usuario.

2.2.2.1 Descripción de los casos de uso

A continuación definen formalmente los casos de uso del servidor.

Caso de uso	Device deferred signature
Actores	Client
Descripción	Este caso de uso describe el proceso de creación de una firma en diferido con el dispositivo cliente.
Precondición	El usuario está autenticado en el servidor y en el servicio de almacenamiento, y los documentos tienen formato PDF.
Postcondición	Se crean los documentos firmados y se suben a la nube.
Flujo básico	1. El caso de uso empieza cuando el servidor recibe la lista de identificadores de documentos a firmar. 2. El servidor se descarga los documentos de la nube. Caso de uso: **Download document**. 3. El servidorcalcula los *digest*de los documentos a firmar. Caso de uso: **Obtain digests.** 4. El servidor se guarda el estado del proceso de firmar documento en memoria, para cuando reciba los datos de las firmas. 5. El **cliente** manda las firmas de los documentos al servidor. 6. El servidor compone los documentos con las firmas. Caso de uso: **Construct signatures**. 7. El servidor sube los documentos firmados a la nube. Caso de uso: **Upload document**. 8. El servidor notifica al **cliente**de que el proceso ha finalizado satisfactoriamente.

Caso de uso	Obtain digests **hereda de** Device deferred signature
Actores	Client
Descripción	Este caso de uso describe el procedimiento para obtener los valores de los *digest* de los documentos a firmar.
Precondición	Los documentos recibidos tienen formato PDF y el usuario está autenticado en el servidor y en el servicio de almacenamiento.
Postcondición	Se calculan los *digest* de los documentos y se crea un nuevo documento con el tamaño estimado que ocupará una vez esté firmado.
Flujo básico	1. El caso de uso empieza cuando el **cliente** manda la lista de documentos para firmar en diferido. 2. Para cada documento, el servidor descarga sus datos de la nube. Caso de uso: **Download document**. 3. El **servidor** crea una copia del documento original, añadiéndole espacio para incrustar la firma una vez esté disponible y calcula el valor del *digest*. 4. El **servidor** envía al **cliente** todos los *digest* junto con los identificadores de los documentos a los que pertenecen.

Caso de uso	Construct signatures **hereda de** Device deferred signature
Actores	Client
Descripción	Este caso de uso describe el proceso donde el servidor compone el documento final con los datos de la firma y crea el documento PDF firmado.
Precondición	Las firmas recibidas tienen un formato válido y el servidor tiene almacenados en disco los documentos con el espacio extra para incrustar la firma.
Postcondición	El servidor compone las firmas en los documentos y crea los documentos PDF firmados. El servidor sube estos documentos al servicio de almacenamiento.
Flujo básico	1. El caso de uso empieza cuando el servidor recibe del **cliente** la lista de firmas con sus identificadores de documento asociados. 2. Para cada firma, el servidor localiza el documento almacenado en disco usando su identificador y lo completa añadiéndole los datos de la firma. 3. El servidor notifica al **cliente** que las firmas se han realizado satisfactoriamente.

Caso de uso	CMS deferred signature
Actores	Client, *TrustedX*
Descripción	Este caso de uso describe como el servidor firma los documentos utilizando un servicio de firma externo del que obtiene un PKCS#7.
Precondición	El usuario está autenticado en el servidor, en el servicio de almacenamiento y en *TrustedX*, y los documentos tienen formato PDF.
Postcondición	Se crean los documentos firmados.
Flujo básico	1. El caso de uso empieza cuando el servidor recibe del **cliente** los identificadores de los documentos a firmar junto con un identificador de certificado de firma perteneciente a *TrustedX*. 2. Para cada documento, el servidor descarga sus datos de la nube. Caso de uso: **Download document**. 3. El **servidor** crea una copia del documento original, añadiéndole espacio para incrustar la firma una vez esté disponible y calcula el valor del *digest*. 4. Para cada digest el servidor hace una llamada a **TrustedX** para que devuelva los datos PKCS#7 de la firma asociada. 4. Una vez finalizadas todas las firmas, para cada una, el servidor localiza el documento almacenado en disco usando su identificador y lo completa añadiéndole los datos de la firma. 5. El servidor notifica al **cliente** que las firmas se han realizado satisfactoriamente.
Flujos alternativos	**Sesión de TrustedX expirada:** Si la sesión de *TrustedX* expira antes del paso 4 del flujo básico, el sistema devuelve un mensaje de error al **cliente** y lo insta a que se autentique de nuevo.

Caso de uso	Login *TrustedX*
Actores	Client, *TrustedX*
Descripción	Este caso de uso describe el proceso de autenticación del cliente al servidor a través de *TrustedX* actuando como un IdP.
Precondición	Ninguna
Postcondición	El servidor autentica al cliente con la identidad proporcionada por *TrustedX*
Flujo básico	1. El caso de uso empieza cuando el servidor recibe del **cliente** el código perteneciente a la autenticación por flujo OAuth 2.0. 2. El servidor manda el código a **TrustedX**. 3. **TrustedX** manda al servidor el *token* de acceso. 4. El servidor manda al **cliente** un mensaje de autenticación satisfactoria.

Caso de uso	Login Google Drive
Actores	Client, Google Drive
Descripción	Este caso de uso explica el procedimiento de como el servidor obtiene los *tokens* OAuth 2.0 de Google Drive.
Precondición	El cliente está autenticado en el servidor
Postcondición	El servidor obtiene las credenciales de Google Drive del cliente.
Flujo básico	1. El caso de uso empieza cuando el servidor recibe del **cliente** el código perteneciente a la autenticación por flujo OAuth 2.0 con Google Drive. 2. El servidor manda el código a **Google Drive**. 3. **Google Drive** manda al servidor las credenciales de acceso al servicio. 4. El servidor manda al **cliente** un mensaje de autenticación satisfactoria.

Caso de uso	Get documents list
Actores	Client, Google Drive
Descripción	Este caso de uso describe como se obtienen la lista de documentos PDF de la aplicación de firma que el cliente almacena en el servicio de almacenamiento en la nube.
Precondición	El cliente está autenticado en el servidor y en el servicio de almacenamiento.
Postcondición	El servidor devuelve al cliente la lista nombres e identificadores de documentos de la aplicación de firma.
Flujo básico	1. El caso de uso empieza cuando el servidor recibe la petición por parte del **cliente** para obtener su lista de documentos. 2. El servidor utiliza las credenciales del usuario para acceder a **Google Drive** y obtener la lista de documentos. 3. El servidor manda al **cliente** su lista de documentos e identificadores.
Flujos alternativos	**No hay documentos**: Si el usuario no tiene documentos el servidor devuelve una lista vacía.

Caso de uso	Download document
Actores	Google Drive
Descripción	Este caso de uso describe como se descarga al servidor un documento del servicio de almacenamiento.
Precondición	El servidor posee las credenciales de Google Drive del cliente.
Postcondición	Se descarga el documento al servidor.
Flujo básico	1. El caso de uso empieza cuando el servidor hace una llamada a **GoogleDrive** pasándole el identificador de documento a descargar y el *token* de acceso. 2. El servidor recibe los datos del fichero y finaliza el caso de uso.
Flujos alternativos	**Error obteniendo el fichero:** Si hay un error descargando el fichero el servidor lo recibe y lo propaga.

Caso de uso	Upload document
Actores	Google Drive
Descripción	Este caso de uso describe como se carga al servicio de almacenamiento un documento del servidor.
Precondición	El servidor posee las credenciales de Google Drive del cliente.
Postcondición	Se carga el documento en Google Drive
Flujo básico	1. El caso de uso empieza cuando el servidor hace una llamada a **Google Drive** pasándole los datos del fichero a subir junto con el *token* de acceso. 2. Cuando la subida finaliza el caso de uso finaliza satisfactoriamente.
Flujos alternativos	**Error subiendo el fichero:** Se puede producir un error si el *token* no es válido en cuyo caso el servidor propagaría el error.

2.3 Diseño de la base de datos

La base de datos tiene que ser capaz de almacenar los usuarios de la aplicación móvil y las credenciales de sus respectivos servicios.

Se ha diseñado siguiendo un modelo relacional permitiendo así representar la información de forma lógica y consistente en tablas relacionadas entre sí.

El diseño del sistema hace que se puedan extraer tres modelos de datos conceptuales a almacenar:

- El usuario de la aplicación
- Credenciales para realizar operaciones en *TrustedX*
- Credenciales para realizar operaciones en Google Drive.

Cada usuario tendrá una credencial para acceder a *TrustedX* y una credencial para acceder a Google Drive. Esto implica que todas tablas de credenciales se tengan que relacionar con la de usuarios a través de claves foranas que permitan la navegabilidad.

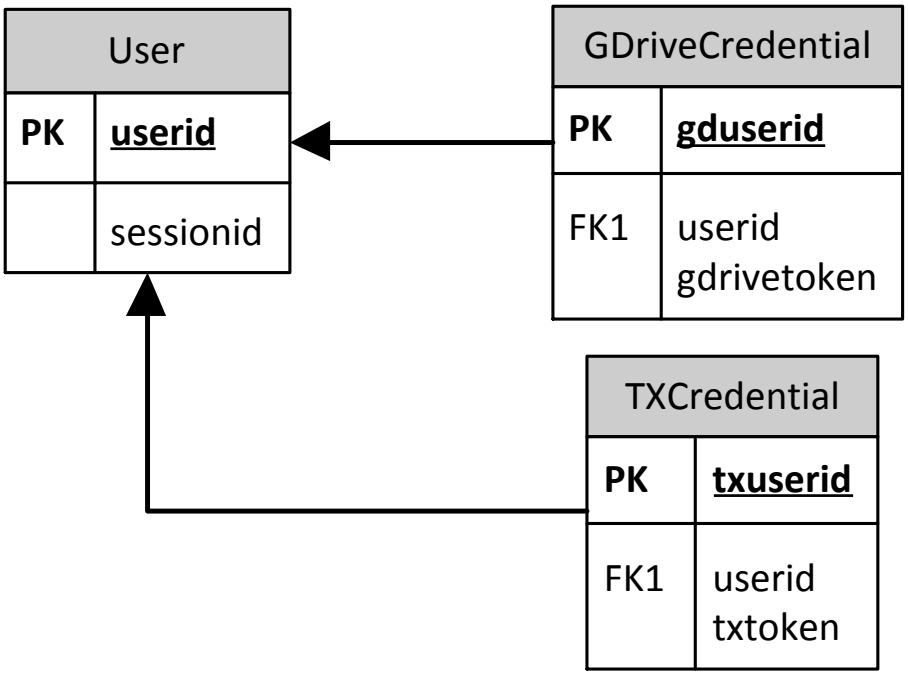


Figura IV.13 Diseño conceptual base de datos

Este diseño permite que el sistema se pueda escalar facilmente incluyendo nuevos servicios (por ejemplo Dropbox), o cambiando los ya existentes (por ejemplo, cambiando *TrustedX* por otro servicio de firma).

Cada usuario tendrá su identificador de usuario, que actuará de clave primária, identificador de sesión y sus credenciales asociadas, que cada una tendrá su identificador de usuario como clave primária y sus *token* que permitirán acceder las funcionalidades de los respectivos servicios.

Los servicios podrán tener más atributos como *refresh tokens* o tiempos de caducidad.

3 Arquitectura general del proyecto

Los componentes del proyecto se relacionan entre sí acorde con el siguiente diagrama de arquitectura.

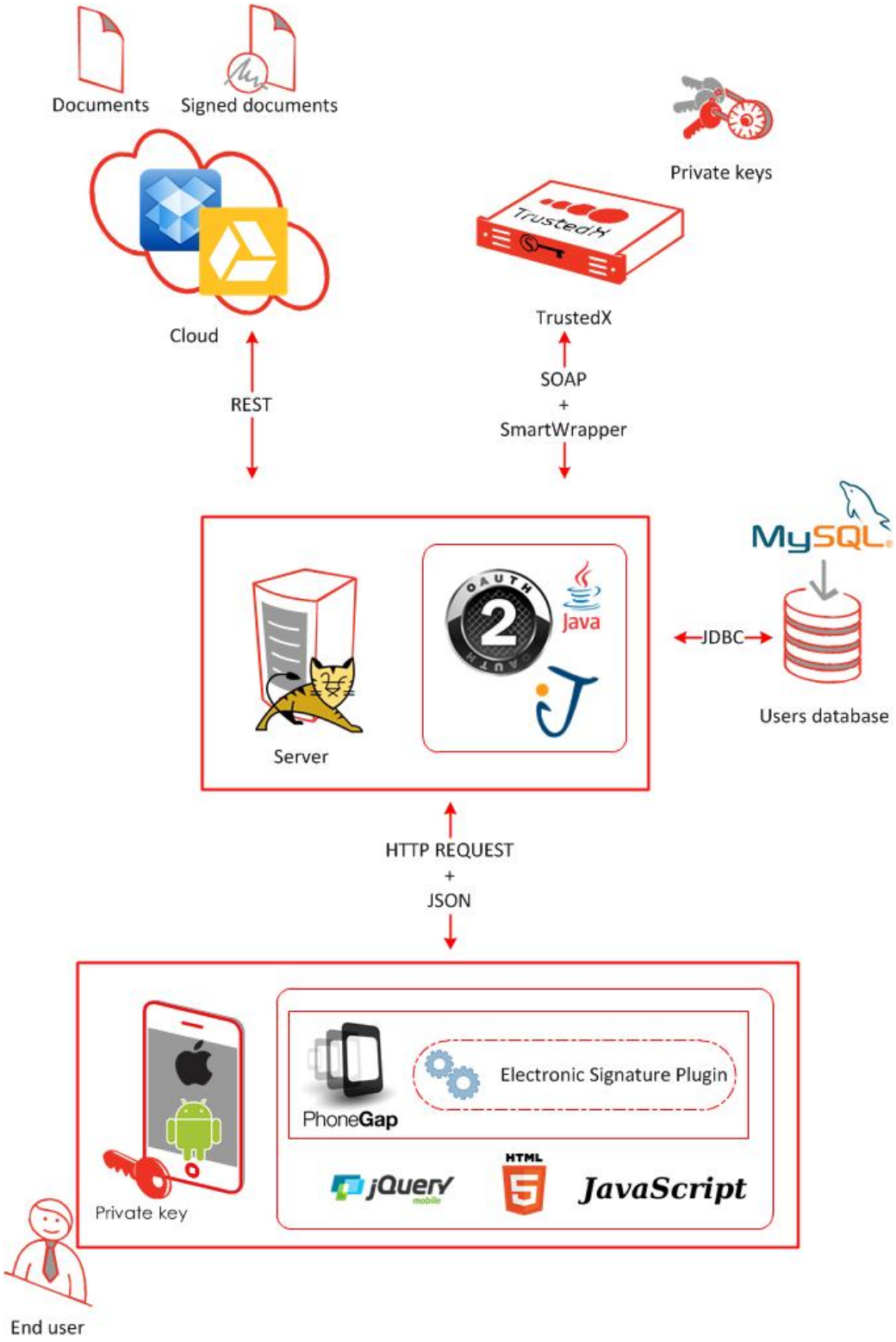


Figura IV.14: Arquitectura general del sistema

El usuario final será quien utilice el dispositivo móvil que incorpora el *framework* con la utilidad de firma.

Este componente se comunica con el servidor Tomcat mediante llamadas HTTP y JSON[12], encapsuladas en un protocolo REST, que a su vez almacena sus datos en una base de datos local MySQL a través de JDBC[13].

Los componentes externos que interactúan con el sistema son los servicios de almacenamiento y *TrustedX*, a los que el servidor accederá en nombre del usuario utilizando el protocolo OAuth 2.0. Estos dos componentes se comunican con el servidor mediante servicios REST y SOAP respectivamente.

Concretamente, en la comunicación entre *TrustedX* y el servidor se utiliza *SmartWrapper*, una API desarrollada por Safelayer para encapsular todas las llamadas SOAP soportadas y simplificar la vida al desarrollador.

Los servicios en la nube almacenarán tanto los documentos firmados como sin firmar del usuario final, y *TrustedX* almacenará las claves privadas del usuario que no estén almacenadas físicamente en el dispositivo móvil.

[12] JavaScript Object Notation, es un format ligero para el intercambio de datos.

[13] Java Database Connectivity, permite la ejecución de operaciones sobre base de datos desde un lenguaje de programación Java.

V - IMPLEMENTACIÓN DEL PROYECTO

1 Introducción

En este capítulo se pondrá en detalle el proceso de implementación del proyecto. Empezando por los entornos de desarrollo donde se ha llevado a cabo, las principales tecnologías que han tomado partido, tanto en el cliente como en el servidor y se mostrarán los diagramas UML más relevantes para comprender mejor la arquitectura interna del sistema.

2 Entornos de desarrollo

Ha sido necesario utilizar dos entornos de desarrollo diferentes para llevar a cabo la implementación del proyecto.

Para la implementación del servidor y de la aplicación Android se ha utilizado Eclipse Java EE IDE for Web Developers versión Indigo Service Release 2 en una máquina Intel Xeon de 64 bits con 12 GB de RAM y sistema operativo Windows 7 Professional Service Pack 1.

Los dispositivos en los que se ha desarrollado para Android han sido un HTC Magic con versión de Android 2.3 y un Samsung Galaxy Nexus con versión de Android 4.2. Este último fue adquirido como requerimiento del proyecto dado que la versión de Android 2.3 del HTC Magic no cumplía con los requisitos necesarios.

Para la implementación de la aplicación iOS se ha utilizado XCode 4.5.2 en una máquina MacBook Pro Intel Core i5 con 8 GB de RAM y sistema operativo Mac OS X Lion 10.7.5.

El dispositivo con el que se ha desarrollado la aplicación iOS ha sido un iPhone 4 con versión de iOS 6.0.

3 Cliente

La parte cliente se compone de una aplicación móvil que actúa de demostrador para los *plugins* desarrollados (que se describirán en secciones posteriores). La aplicación es multiplataforma, aunque los *plugins* se han desarrollado para únicamente tener compatibilidad con los sistemas Android e iOS.

3.1 Plataformas móviles sobre las que se ha implementado

En esta sección se describen los sistemas operativos para las plataformas móviles en las que se ha centrado el proyecto.

3.1.1 Android

Android es un sistema operativo basado en Linux y diseñado principalmente para dispositivos con **pantalla táctil** como teléfonos inteligentes o tabletas. Inicialmente fue desarrollado por la empresa **Android Inc.** con el soporte económico de Google, quien posteriormente compró la empresa en 2005. El primer teléfono comercializado con sistema Android fue lanzado en octubre de 2008.

El sistema operativo se caracteriza por ser *open source* con el código liberado bajo la licencia Apache. Esto permite que el código fuente sea modificado tanto por empresas distribuidoras como por desarrolladores aficionados.

Tiene una gran comunidad de desarrolladores creando aplicaciones, también denominadas *apps*. A finales de octubre de 2012 llegan al número de 700.000 en la tienda de aplicaciones de Android, llamada *Google Play*. Éstas se desarrollan con una versión personalizada del lenguaje de programación *Java*.

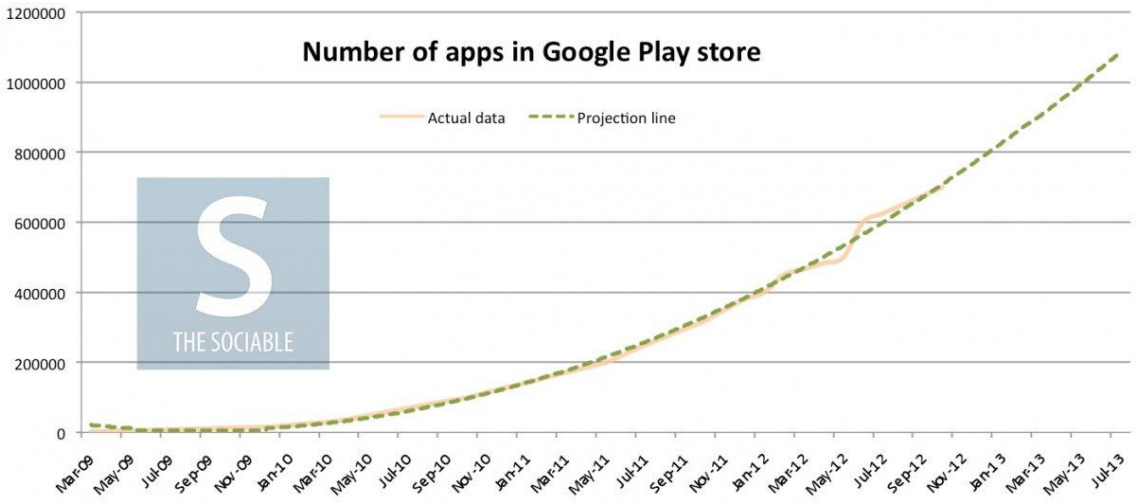


Figura V.1 Evolución de apps en Google Play. Fuente: http://sociable.co

El *kernel* de Android 4.0 *Ice Cream Sandwich* en adelante se basa en el *kernel* 3.x de Linux, con un *middleware*, librerías y API escritos en C. Los *frameworks* de

aplicaciones incluyen librerías compatibles con Java basados en *Apache Harmony*. Android usa la máquina virtual *Dalvik* con compilación *just-in-time*[14] para ejecutar el código ejecutable de *Dalvikdex-code* que normalmente es traducido al *bytecode* de *Java*. La principal arquitectura utilizada es ARM[15].

Android es el sistema operativo para dispositivos móviles más utilizado a nivel global y la elección de muchas compañías tecnológicas que requieren sistemas personalizables, ligeros y de bajo coste para dispositivos de alta tecnología sin tener que desarrollar los suyos propios desde cero. A pesar de ser desarrollado como sistema para teléfonos inteligentes, actualmente podemos verlo en otra tipología de dispositivos, como por ejemplo televisores o videoconsolas. Actualmente hay más de 500 millones de dispositivos Android activados.

3.1.2 iOS

iOS es un sistema operativo para dispositivos móviles desarrollado y distribuido por la empresa **Apple Inc.** Fue lanzado originalmente en enero 2007 para los dispositivos de Apple iPhone e iPod Touch aunque no permitía aplicaciones de terceros. No sería hasta febrero de 2008 que Apple liberaría un SDK público para que desarrolladores externos pudieran crear sus propias aplicaciones. Al contrario que otros sistemas operativos posteriores como Windows Phone o Android, la instalación de iOS está restringida para terminales que no sean manufacturados por Apple.

Lo que caracterizó al sistema en su lanzamiento fue que su interfaz de usuario se basaba en el concepto de **manipulación directa**, donde los usuarios pueden interactuar directamente con los elementos en pantalla mediante gestos como pulsaciones, deslizamientos o pellizcos. Los dispositivos iOS cuentan además con acelerómetros que permiten detectar cuando el teléfono se sacude o cambia su orientación. Esto permite controlar diferentes acciones simplemente con el movimiento del terminal en un espacio 3D.

---

[14] Método para mejorar el rendimiento de los programas basados en *byte code* en tiempo de ejecución.

[15] Advanced RISC Machine.

iOS deriva de OS X[16] con quien comparte la *Darwin foundation*. El sistema tiene cuatro niveles de abstracción: *Core OS*, *Core Services*, *Media* y *Cocoa Touch*.

El SDK[17] para desarrollar en iOS, al igual que para OS X, es XCode, y el lenguaje de programación nativo es Objective-C. Las aplicaciones son publicadas en la tienda *App Store* con un estricto filtro por parte de Apple. Al contrario que Android, Apple solo permite la instalación de aplicaciones a través de su tienda oficial que cuenta con cerca de 800.000 aplicaciones.

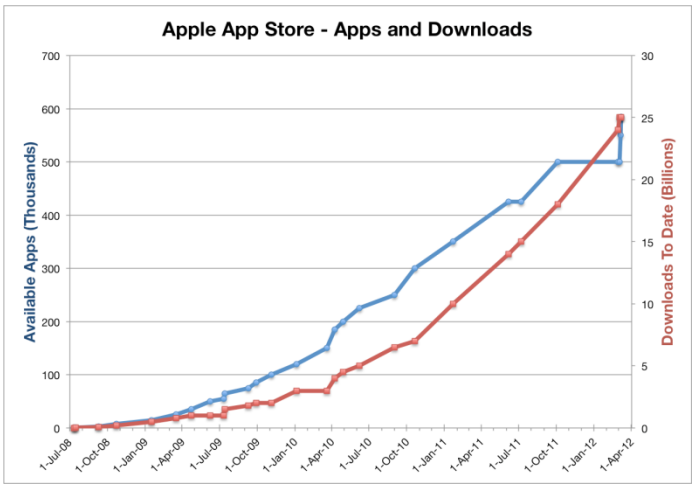


Figura V.2 Evolución de apps en App Store. Datos extraídos de Apple.

En septiembre de 2012 se habían vendido cerca de 400 millones de dispositivos iOS en todo el mundo.

[16] Sistema operativo gráfico basado en Unix desarrollado por Apple para ordenadores Mac.

[17] System Development Kit.

3.2 *Tecnologías utilizadas*

Esta sección describe las diferentes tecnologías que se han utilizado en la implementación del cliente para las plataformas móviles iOS y Android.

3.2.1 *Framework* PhoneGap

La principal tecnología que ha permitido desarrollar el cliente móvil para un ecosistema multiplataforma ha sido PhoneGap. El *framework*, descrito en capítulos anteriores, permite desarrollar aplicaciones híbridas de manera que, a pesar de estar desarrollando una aplicación web, se pueda acceder a la mayoría de recursos nativos del teléfono (sensores, cámara, almacenamiento, etc.).

La arquitectura de PhoneGap, tal y como se aprecia en la siguiente figura, consiste en código web ejecutado por el motor interno del sistema, es decir, un navegador integrado. Éste código web ejecuta *plugins* nativos del sistema a través de una API Javascript. Este código nativo llama a la API del sistema operativo específico de la plataforma para acceder a sus recursos específicos (sensores, graficos, etc).

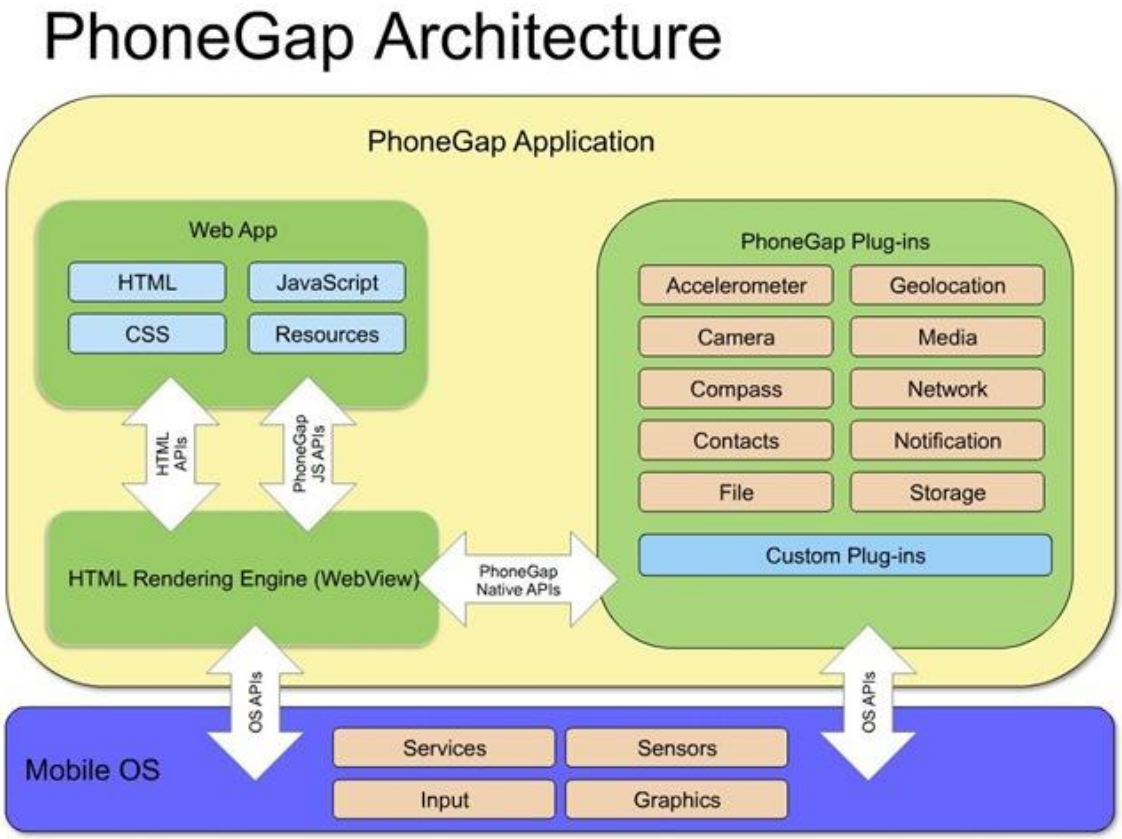


Figura V.3: Arquitectura de PhoneGap

Ésta API presenta un problema: **limita** las funcionalidades que puede tener la aplicación, dado que no todos los recursos nativos del dispositivo están disponibles: únicamente los más populares. Tampoco hay compatibilidad de la API con todas las plataformas habiendo algunas que soportan más funcionalidades que otras, siendo Android e iOS las que muestran un 100% de compatibilidad.

	iOS iPhone / iPhone 3G	iOS iPhone 3GS and newer	Android	OS 5.x	OS 6.0+	WebOS	WP7	Symbian	Bada
ACCELEROMETER	✓	✓	✓	✓	✓	✓	✓	✓	✓
CAMERA	✓	✓	✓	✓	✓	✓	✓	✓	✓
COMPASS	✗	✓	✓	✗	✗	✗	✓	✗	✓
CONTACTS	✓	✓	✓	✓	✓	✗	✓	✓	✓
FILE	✓	✓	✓	✓	✓	✗	✓	✗	✗
GEOLOCATION	✓	✓	✓	✓	✓	✓	✓	✓	✓
MEDIA	✓	✓	✓	✗	✗	✗	✓	✗	✗
NETWORK	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (ALERT)	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (SOUND)	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (VIBRATION)	✓	✓	✓	✓	✓	✓	✓	✓	✓
STORAGE	✓	✓	✓	✓	✓	✓	✓	✓	✗

Figura V.4: Funcionalidades de PhoneGap en diferentes plataformas

3.2.1.1 Desarrollo de plugins en PhoneGap

La limitación de PhoneGap en cuanto al acceso a funcionalidades nativas soluciona ofreciendo a los desarrolladores la capacidad de desarrollar sus **propios plugins**, permitiendo así, extender las funcionalidades del *framework* adaptándolo a sus necesidades.

El *plugin* tiene que estar implementado en código nativo, y la clase principal tiene que heredar de una clase de las librerias de PhoneGap. Después hay que añadir la definición del *plugin* en un fichero de configuración del proyecto y crear la API Javascript que encapsula la llamada del *plugin*.

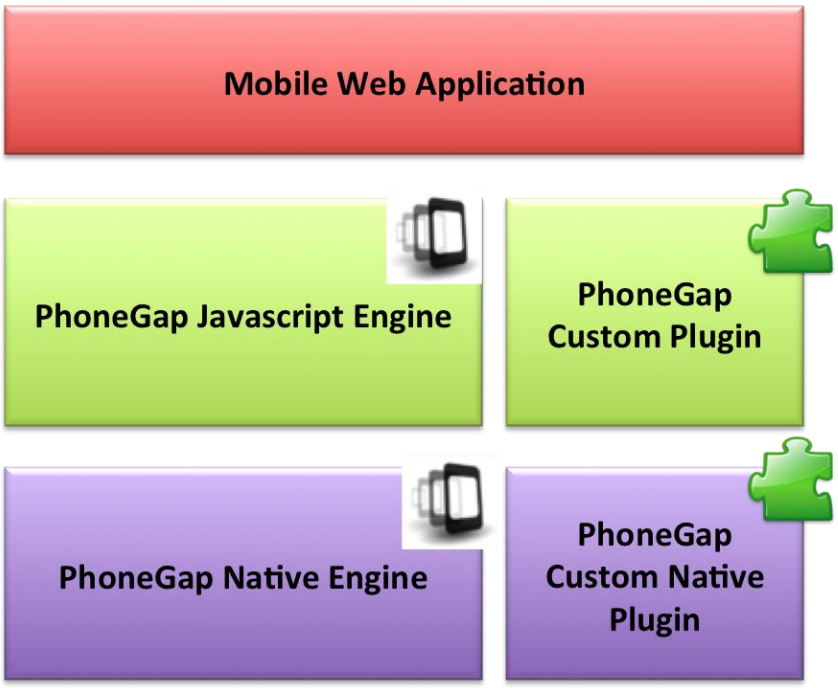


Figura V.5: Arquitectura de los *plugins* en PhoneGap

A pesar de que crear un *plugin* nativo va en contra del principio multiplataforma puede ser interesante si necesitamos acceder a determinadas funcionalidades no implementadas en la API de PhoneGap, o si queremos realizar tareas pesadas que resultarían inviables de ejecutar en JavaScript.

3.2.2 Tecnologías web

Para implementar la capa multiplataforma de los clientes móviles ha sido necesario utilizar las tecnologías web que se describen en este apartado.

3.2.2.1 HTML

HTML es el lenguaje principal con el que se construyen las páginas web. Las siglas significan *HyperText Markup Language* o Lenguaje de Marcado de HiperTexto.

El HTML se escribe en forma de etiquetas y puede describir la estructura principal de un documento. También puede incluir *scripts* que afecten el comportamiento de los navegadores web u otros intérpretes.

3.2.2.2 CSS

El *Cascading Style Sheet* (CSS), o Hoja de Estilo en Cascada, es un lenguaje de hoja de estilo usado para definir las semánticas de presentación (es aspecto y formateo) de un documento escrito en un lenguaje de marcado, como HTML.

3.2.2.3 JavaScript

La principal tecnología web que ha habido que utilizar y que conforma toda la lógica principal del cliente móvil es JavaScript.

JavaScript es un lenguaje de programación interpretado con propiedades de la programación orientada a objetos (POO). La sintaxis es parecida a la de C++, C y Java, sin embargo esta similitud termina en el ámbito sintáctico.

No existe tipado en JavaScript, es decir, las variables no necesitan ser especificadas de un tipo u otro. Es el propio intérprete quien se encarga, en tiempo de ejecución, de asignar tipos a las variables.

JavaScript se usa comúnmente en navegadores web y, en ese contexto, el núcleo de propósito general se extiende con objetos que permiten a los *scripts* interactuar con el usuario, controlar el navegador web y alterar el contenido del documento web. Ésta versión incrustada de JavaScript ejecuta *scripts* incrustados dentro del código HTML de las páginas web. Comúnmente se denomina *client-side* JavaScript para resaltar que los *scripts* son ejecutados en la máquina cliente en lugar de en el servidor web.

3.2.2.4 JQuery y JQuery Mobile

JQuery es una biblioteca de JavaScript multi-navegador que simplifica la manera de interactuar con los documentos HTML. Es la librería JavaScript más popular a día de hoy.

JQuery es gratuito y de código abierto. Su sintaxis está diseñada para simplificar la navegación por el documento, seleccionar objetos del DOM[18], crear animaciones, manejar eventos y desarrollar aplicaciones Ajax[19]. JQuery también capacita a los desarrolladores crear sus propios *plugins* encima de la

[18] Document Object Model

[19] Asynchronous JavaScript and XML

librería JavaScript. Esta visión modular de la librería JQuery permite la creación de páginas web dinámicas muy potentes y de aplicaciones web.

JQuery Mobile es la versión de JQuery orientada a dispositivos con pantalla táctil. Permite controlar la mayoría de interacciones que un usuario puede hacer con un teléfono inteligente o una tableta, gestos como tocar o deslizar, entre otros. JQuery Mobile es compatible con otros *frameworks* móviles como el anteriormente descrito PhoneGap.

JQuery Mobile también permite la creación de temas para las aplicaciones mediante un potente *framework* que ofrece a los desarrolladores la posibilidad de personalizar colores y ciertos aspectos del CSS de las funcionalidades de la interfaz de usuario.

Los desarrolladores pueden usar la herramienta Theme Roller (herramienta que se ha usado para crear el formulario de autenticación OAuth 2.0 de *TrustedX*) para personalizar los aspectos de la interfaz y descargar luego un fichero CSS con la configuración gráfica para cargarlo en su aplicación.

3.2.3 Tecnologías nativas

3.2.3.1 Java

Java es el lenguaje con el que se desarrolla nativamente en sistemas Android. Es un lenguaje de alto nivel, de propósito general, concurrente, basado en clases y orientado a objetos que esta específicamente diseñado para tener el menor número de dependencias de implementación posible. Se pretende que los desarrolladores escriban el código una vez y lo ejecuten en cualquier sitio, por lo que el código que se ejecuta en una plataforma, no necesita ser recompilado para ejecutarse en otra.

Las aplicaciones Java son compiladas en *bytecode*, que puede ser ejecutado únicamente en una máquina virtual de Java (JVM) independientemente de la arquitectura del sistema.

La sintaxis del lenguaje deriva mucho de C y C++, pero tiene muchas menos funcionalidades de bajo nivel. Por ejemplo, no se permite el acceso directo a memoria. Esto permite disminuir el riesgo de errores por parte del desarrollador.

El propio lenguaje se encarga de la gestión de memoria del ciclo de vida de los objetos. El programador determina cuando un objeto es creado y el motor de Java se encarga de liberar la memoria una vez estos ya no están en uso.

Java es uno de los lenguajes de programación más populares actualmente, particularmente para aplicaciones cliente-servidor, con 10 millones de usuarios reportados.

3.2.3.2 Objective-C

Objective-C es el lenguaje de programación con el que se desarrollan aplicaciones nativas en los sistemas de Apple, como OSX e iOS. Es de propósito general, alto nivel y orientado a objetos. Añade la tipología de los mensajes de Smalltalk al lenguaje de programación C.

Su sintaxis se compone de una fina capa encima de C, más específicamente, es un superconjunto de C. Es técnicamente posible compilar cualquier programa en C con un compilador de Objective C, e incluir libremente cualquier código C dentro de una clase Objective-C.

Objecive-C deriva su sintaxis de Smalltalk. Toda la sintaxis para las operaciones no orientadas a objetos (incluyendo variables primitivas, preprocesado, expresiones, declaraciones de funciones y llamadas) son idénticas a C, mientras que la sintaxis para las funcionalidades de POO son una implementación del estilo de Smalltalk.

3.3 Implementación de la aplicación móvil

3.3.1 Configuración del entorno PhoneGap

Desarrollar una aplicación mediante el *framework* PhoneGap requiere de una configuración previa del proyecto en el que se va a trabajar. El proyecto tiene que incluir las librerías adecuadas y hay que crear la clase **nativa** principal que será la que cargará en **intérprete web** que ejecutará el código del desarrollador.

Estos pasos se pueden realizar "a mano" o por el contrario, para agilizar el proceso de crear clases, importar librerías y modificar archivos de configuración, desde el sitio web de PhoneGap se pueden descargar varios *scripts*(uno para cada plataforma sobre la que vayamos a desarrollar) que se encargan de realizar estas tareas.

El desarrollador únicamente tiene que ejecutar el *script* (un archivo .bat en Windows y un .sh en máquinas Unix) pasando como parámetro datos sobre el proyecto, como su nombre y el paquete principal.

```
./create <project_folder_path><package_name><project_name>
```

Esto crea un directorio con el proyecto ya creado y configurado para el IDE con el que trabajemos. En el caso de Android crearía un proyecto de Eclipse y en el caso de iOS crearía un proyecto de XCode.

En ambos casos el nuevo proyecto contiene una aplicación demo ejecutable por el dispositivo.

3.3.2 Implementación de la aplicación JavaScript

El código principal con el que se ha implementado la aplicación móvil es JavaScript, de manera que se pueda ejecutar en cualquier plataforma.

Para simplificar el desarrollo se ha hecho uso de las librerías JQuery y JQuery Mobile, utilizando también el CSS proporcionado por JQuery Mobile para dar un aspecto a la interfaz gráfica más similar al que encontraríamos en una aplicación móvil nativa.

Estas librerías también ofrecen métodos para realizar llamadas AJAX al servidor, pudiendo así mantener una comunicación asíncrona sin interferir con la visualización o el comportamiento de la aplicación. Todas las llamadas al servidor, por ejemplo obtener los documentos, se hacen mediante AJAX.

3.3.3 Implementación de la autenticación OAuth 2.0 con TrustedX

Para realizar la autenticación OAuth 2.0 hay que bajarse un formulario web del servidor que permita al usuario rellenar con sus datos (usuario y contraseña) y mandarlo al servicio correspondiente.

Esto se ha implementado con la inclusión de un navegador integrado en la aplicación JavaScript utilizando uno de los *plugins* incluidos en la librería de PhoneGap. A continuación se dan los detalles de cómo se aplica el flujo de autenticación OAuth 2.0 en la aplicación.

3.3.3.1 Aplicación del flujo OAuth 2.0

Para que la aplicación móvil pueda realizar tareas en *TrustedX* en nombre del usuario es necesario el *token* OAuth 2.0. En esta sección se detalla la implementación del flujo.

Se trata de una autenticación basada en el flujo *Authorization Code*, como la explicada en la sección 2 del capítulo CONCEPTOS BÁSICOS, con una autorización implícita. Esto es, no se le pide explícitamente al usuario si autoriza que el servidor intermedio utilice *TrustedX* para firmar en su nombre.

En el siguiente esquema se observa el funcionamiento del proceso de autenticación.

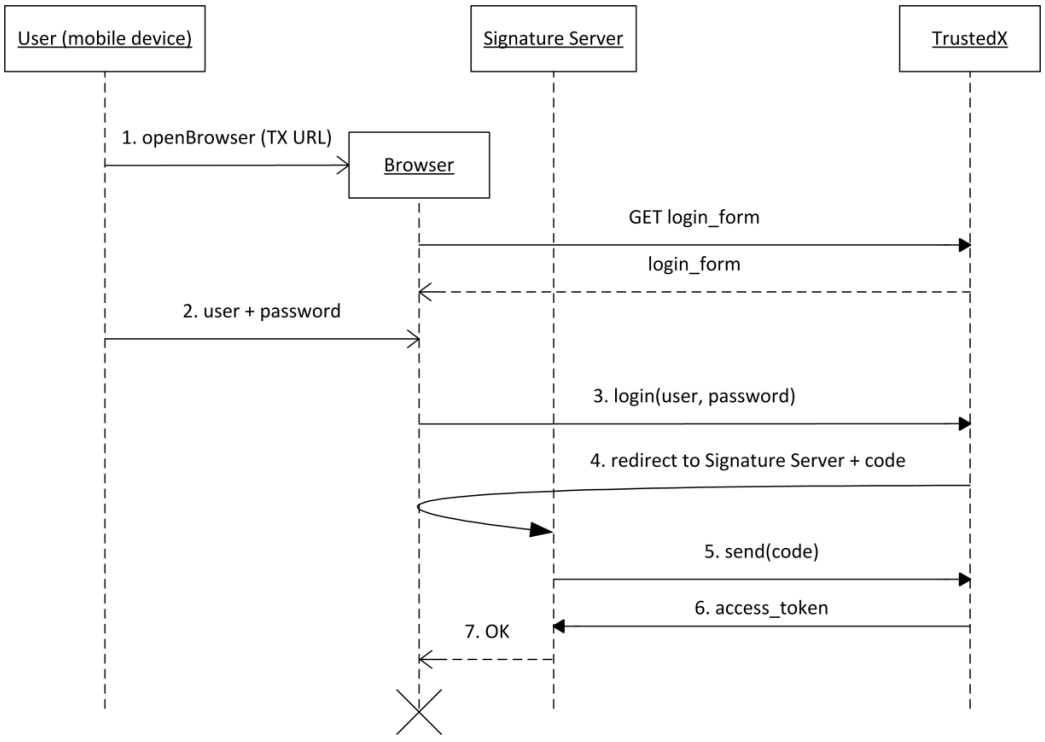


Figura V.6: Aplicación del flujo OAuth 2.0

1. La aplicación abre un navegador contra una URL que muestra el formulario de autenticación (usuario y contraseña) de *TrustedX*. Especificando también la URL del servicio que va a actuar en su nombre, en este caso, el *Signature Server*, así como el *client id* para identificar qué aplicación es la que quiere acceder.

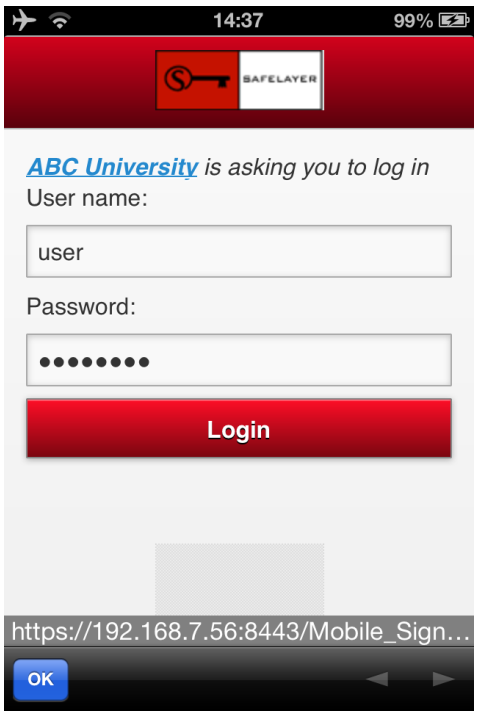


Figura V.7: Formulario de autenticación móvil

2. El usuario introduce su nombre de usuario y contraseña desde su dispositivo móvil.
3. El navegador manda los campos de autenticación a *TrustedX* y éste los valida.
4. *TrustedX* manda el *code* al navegador del dispositivo junto con una orden de redirección a la URL del *Signature Server* especificada.
5. Cuando el *Signature Server* recibe el *code* lo manda de vuelta a *TrustedX* mediante una solicitud autenticada de token. *TrustedX* valida la identidad del *Signature Server* a partir de su *client id* y *client secret* y comprueba que el *code* es el mismo que ha mandado él y que viene de la misma URL que se ha especificado en el punto 1.
6. Si todo está correcto manda el *access token* al *Signature Server*.

7. Una vez completado el proceso se cierra el navegador y el *Signature Server* ya puede utilizar *TrustedX* para firmar en nombre del usuario, utilizando el *access token* en las peticiones a los servicios.

El uso de *PhoneGap* permite simplificar mucho el proceso de autenticación, que se realiza con la siguiente función Javascript.

```
1  function oauthLogin() {
2
3      var ref = window.open("https://trustedx/adaptive/adaptive.jsp?
           client_id=demord&redirect_uri=https://signature-server/
           txauth&response_type=code", '_blank', 'location=yes');
4
5      ref.addEventListener('loadstop', function(e) {
6
7          var loc = e.url;
8
9          if (loc.indexOf("https://signature-server/acceso.jsp") > -1) {
10
11             alert("Login successful!");
12             ref.close();
13         }
14
15     });
16 }
```

Figura V.8: JavaScript para realizar la autenticación OAuth 2.0

La aplicación corre en el dispositivo (no en el navegador) por lo que primero hay que lanzar un navegador web contra el servicio de autenticación OAuth 2.0 de *TrustedX (línea 3).*

Este navegador corre asíncronamente y hay que poder capturar las redirecciones que sufre. Esto es un requisito en la concesión *Authorization Code* según el RFC. Para ello, se registra un *callback* en el navegador que se llamará cada vez que una página termine de cargar *(línea 5*).

En cuanto se detecte la página de destino después de una autenticación exitosa (que es conocida por el desarrollador) *(línea 9)* la autenticación ha terminado y se puede cerrar el navegador *(línea 12*) para volver a la aplicación. La aplicación puede aprovechar los atributos de identidad que contiene la respuesta de autenticación.

3.3.3.2 Ejemplo de caso de uso: firmar documento

En esta sección se dará un ejemplo de uso del *access token* obtenido en el proceso de autenticación OAuth 2.0.

El caso de uso consiste en firmar un documento, controlado por el servidor, con *TrustedX* como servicio de firma. En el siguiente esquema se muestra su flujo de ejecución simplificado.

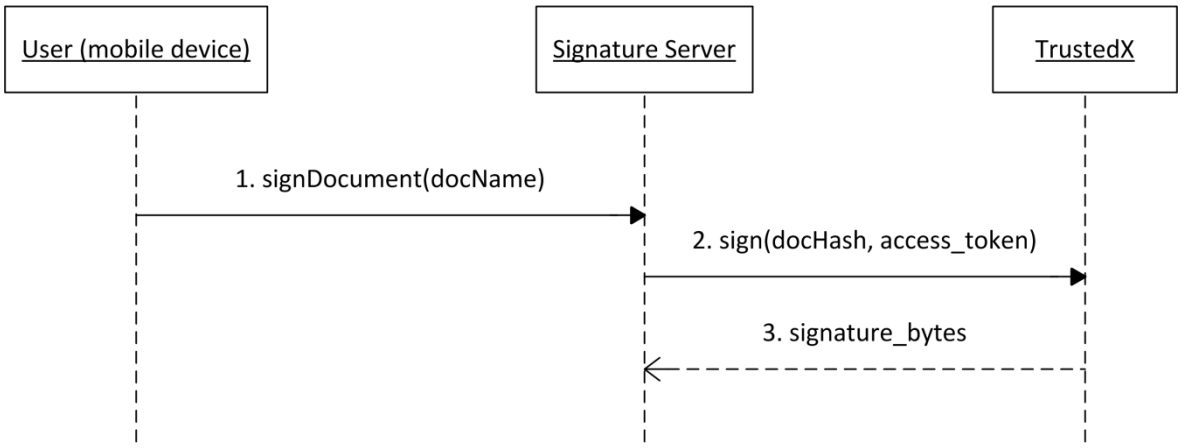


Figura V.9: Caso de uso: firmar documento

1. El usuario desde su dispositivo selecciona el nombre del documento que quiere firmar y manda la orden al *Signature Server*.
2. El *Signature Server* calcula el hash del documento y lo manda a *TrustedX* para que calcule la firma en nombre del usuario junto con el *access token* obtenido en el proceso de autenticación.
3. *TrustedX* verifica el *access token*, calcula la firma y devuelve los bytes al *Signature Server* que compone la firma en el documento.

4 Servidor

Esta sección define los detalles sobre la implementación del servidor intermedio del proyecto, cuyas funcionalidades ya se han descrito en el capítulo IV.

Se describen las principales tecnologías implicadas en su desarrollo, una definición de la interfaz REST del servicio, el diagrama de clases general del sistema y una descripción del comportamiento del flujo de firma.

4.1 Tecnologías utilizadas

Este apartado describe las principales tecnologías que se han aplicado durante la implementación del servidor.

4.1.1 Java EE

Java Enterprise Edition, traducido informalmente como Java Empresarial, es una plataforma de programación que forma parte de la plataforma Java. La plataforma ofrece una API y un entorno de ejecución para desarrollar y ejecutar software empresarial, incluyendo servicios web y otras aplicaciones en red a gran escala, multicapa, escalables, fiables y seguras. Java EE hereda de la plataforma Java, Stantard Edition, ofreciendo una API para mapeo de objetos relacionales, arquitecturas multicapa distribuidas y servicios web. La plataforma incorpora un diseño basado mayormente en componentes modulares que se ejecutan en un servidor de aplicaciones. El software desarrollado para Java EE esta mayormente desarrollado mediante el lenguaje de programación Java.

4.1.2 Servicios web

Un servicio web es un sistema software diseñado para soportar interacciones de máquina a máquina sobre una red. Tiene una interfaz descrita en un formato procesable por una máquina (concretamente WSDL[20]). Los otros sistemas interactúan con el servicio web de la manera que se especifica en su descripción usando mensajes SOAP[21], típicamente transportados usando HTTP con una serialización en XML.

[20] Web Services Description Language

[21] Simple Object Access Protocol

Existen maneras de simplificar estas comunicaciones, prescindiendo de las definiciones de los ficheros WSDL y los mensajes con datos en XML. Una de ellas es utilizando las técnicas REST[22] que permiten definir una interfaz simple sobre la que hacer las llamadas a los servicios.

En este proyecto se han usado tanto los servicios web de tipo SOAP (para realizar las llamadas al servicio *TrustedX*) como de tipo REST (para crear la interfaz del servidor y acceder a Google Drive).

4.1.3 iText

iText es una librería de código abierto gratuita para crear y manipular ficheros PDF. Permite al desarrollador utilizar las siguientes funcionalidades:

- Servir un PDF a un navegador.
- Crear documentos dinámicos a partir de un fichero XML o una base de datos.
- Usar funcionalidades interactivas de los PDF.
- Añadir marcadores de página, número de página, marcas de agua, códigos de barras, etc.
- Separar, concatenar y manipular páginas PDF.
- Rellenar automáticamente formularios PDF.
- Añadir firmas digitales a un fichero PDF.

Es por esta última característica que ha sido necesario incluir la librería iText en el proyecto ya que, de otro modo, la tarea de incluir las firmas en los documentos PDF hubiese resultado mucho más pesada.

4.1.4 MySQL

Para implementar la persistencia de los datos de usuarios que el servidor tiene que almacenar se ha decidido utilizar la base de datos MySQL.

MySQL permite la creación de bases de datos relacionales gráficamente mediante su herramienta MySQL Workbech. El usuario puede diseñar las

[22] Representational State Transfer

tablas y sus relaciones en un entorno gráfico y luego exportar un *script* que crea las instancias de la base de datos diseñada.

A su vez MySQL es una de las bases de datos más usadas entre los desarrolladores por su fiabilidad manejando grandes cantidades de datos y consultas.

4.1.5 Apache Tomcat

Apache Tomcat, también llamado Jakarta Tomcat, es un contenedor de servlets desarrollado en el proyecto Jakarta por la Apache Software Foundation. Actúa como servidor web y soporta servlets y JavaServer Pages (JSP). Está escrito en Java, por lo que se puede ejecutar en cualquier máquina virtual de Java (JVM).

Para el desarrollo de este proyecto se ha utilizado Apache Tomcat en su séptima versión para albergar el código del servidor.

4.2 Implementación del servicio REST

Para facilitar el acceso a las funcionalidades del servidor desde los dispositivos móviles, se ha definido una interfaz de tipo REST con la siguiente especificación:

Método	Listar certificados de *TrustedX*
Descripción	Lista los certificados de *TrustedX* asociados al usuario
Llamada	`GET /restapi/list/signingcerts`
Parámetros	• `sessionid`: identificador de sesión del usuario.
Resultado	Vector de objetos JSON conteniendo el nombre del certificado y su identificador: `JSONArray [{certname: string, certid: string}]`

Método	Realizar firma CMS
Descripción	Realiza una firma utilizando el servicio *TrustedX*.
Llamada	`POST /restapi/sign/cms`
Parámetros	• `sessionid`: identificador de sesión del usuario. • `names`: vector de objetos JSON con los nombres e identificadores de los documentos a firmar: `JSONArray [{doc: string}]` • `certid`: identificador del certificado con el que realizar la firma.
Resultado	Objeto JSON con información sobre el resultado de la operación: `{result: string}`

Método	Realizar firma diferida: obtener *digests*
Descripción	Para realizar la firma diferida, es decir, firmar en el dispositivo móvil, pero calcular el *digest* y montar la firma en el servidor, es necesario primero obtener el valor del *digest* con éste método.
Llamada	`POST /restapi/sign/deferred/digest`
Parámetros	• `sessionid`: identificador de sesión del usuario. • `chain`: vector de objetos JSON con el certificado para realizar la firma y su cadena de certificados asociada: `JSONArray [{doc: string, digest: b64}]`
Resultado	Vector de objetos JSON con los identificadores de los documentos y sus valores de *digest* codificados en Base64: `JSONArray [{doc: string, digest: b64}]`

Método	Finalizar firma diferida: construir documentos firmados
Descripción	Cuando se está realizando la firma diferida, calculando el valor de la firma en el dispositivo móvil, una vez se tiene ese valor, se llama a este método para que el servidor componga los documentos firmados.
Llamada	`POST /restapi/sign/deferred/construct`
Parámetros	• sessionid: identificador de session del usuario. • signatures: vector de objetos JSON con los identificadores de documento y sus firmas asociadas: `JSONArray [{doc: string, signature: b64}]`
Resultado	Objeto JSON con información sobre el resultado de la operación: `{result: string}`

4.3 Diagrama de clases del servidor

En este apartado se mostrará la arquitectura interna de las clases del servidor a nivel conceptual. Para facilitar su comprensión el sistema se dividirá en subsistemas, cada uno de ellos explicado de forma separada.

La arquitectura se ha diseñado pensando en un sistema *multi-tier*, es decir, varias capas para separar la lógica del acceso a datos. La estructura de los principales componentes es totalmente escalable para que añadir funcionalidades no cree malfuncionamientos en otras zonas del código.

4.3.1 Núcleo del sistema

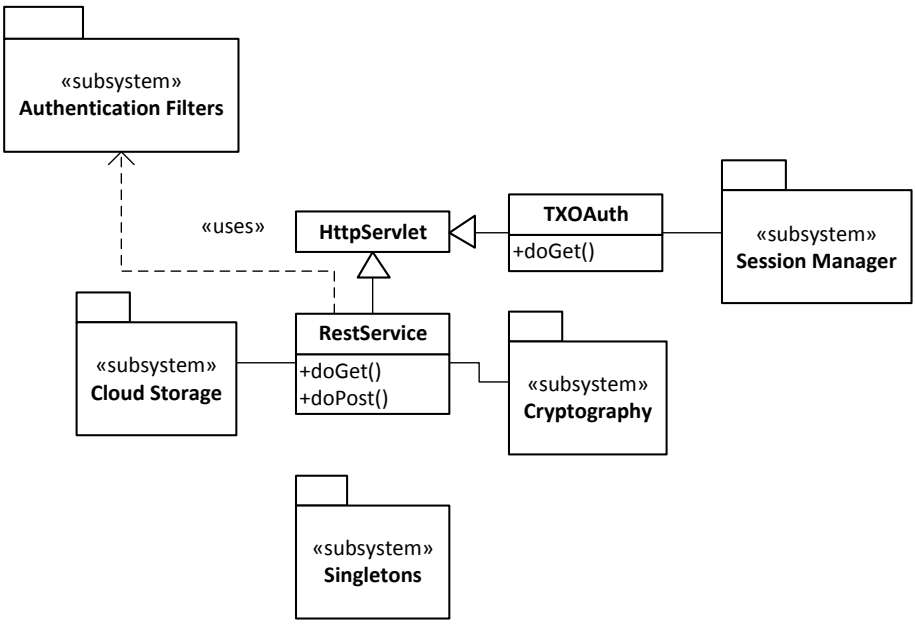


Figura V.10: Núcleo de clases del servidor

En este diagrama se puede ver la arquitectura de clases general del servidor. Al implementar un servicio REST el punto de entrada principal es la clase de Java *HttpServlet*. De esta clase heredan las dos siguientes:

- *TXOAuth*: Clase encargada de realizar la autenticación OAuth 2.0 con TrustedX y generar los identificadores de sesión con el subsistema *Session Manager*.

- *RestService*: Una vez realizada la autenticación, está clase se encarga de gestionar todas las llamadas a servicios REST. Para verificar que el usuario está autenticado se usa el subsistema *Authentication Filters*. A su vez todas las operaciones criptográficas se encapsulan en el subsistema *Cyptography* y las operaciones relacionadas con la gestión de ficheros en la nube en el subsistema *Cloud Storage*.

El subsistema *Singletons* contiene las clases que siguen el patrón *Singleton* para dar funcionalidades accesibles desde cualquier parte del código.

4.3.2 Subsistema *Authentication Filters*

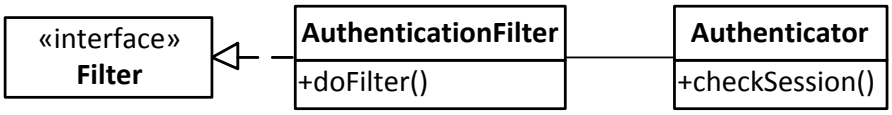


Figura V.11: Subsistema *Authentication Filters*

Este subsistema es el encargado de filtrar todas las peticiones que se hacen a la API REST para verificar que el usuario que la hace está autenticado correctamente.

- *AuthenticationFilter*: Clase principal del filtro, implementa la interfaz de Java *Filter* usada por el servidor *Tomcat*. El método *doFilter* es el encargado de llamar a los métodos de verificación.

- *Authenticator*: En esta clase se realiza la comprobación de las credenciales de usuario a partir del identificador de sesión, en el método *checkSession*.

4.3.3 Subsistema *Session Manager*

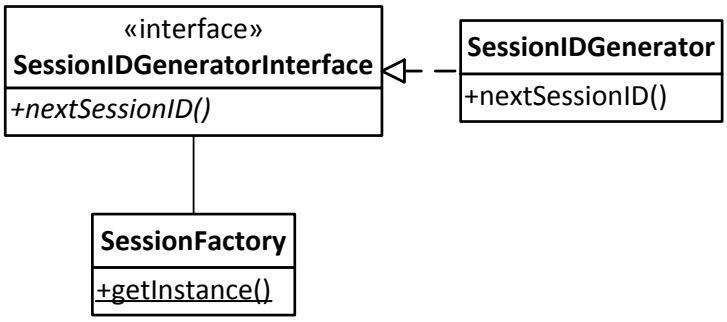


Figura V.12: Subsistema *Session Manager*

Session Manager se encarga de generar los identificadores de sesión. La estructura de clases sigue un modelo *Factory Pattern* para facilitar la escalabilidad en caso de querer añadir otros métodos de generación del identificador.

- *SessionFactory*: Clase que se encarga de generar las instancias de la interfaz *SessionIDGeneratorInterface* a través del método estático *getInstance*.

- *SessionIDGeneratorInterface*: Todas las clases generadoras de identificadores de sesión implementan esta interfaz.

- *SessionIDGenerator*: Implementación de la interfaz anterior. El método *nextSessionID* se encarga de generar la cadena de texto del identificador de sesión a partir de un tamaño.

4.3.4 Subsistema *Cloud Storage*

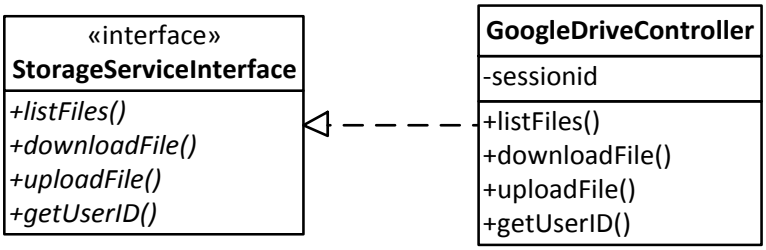


Figura V.13: Subsistema *Cloud Storage*

Toda la gestión de ficheros en la nube se realiza en este subsistema. Desde otras zonas del código siempre se referencia a la interfaz para facilitar la escalabilidad en caso de añadir otros servicios aparte de *Google Drive* (p.e. *Dropbox*).

- *StorageServiceInterface*: Interfaz que define el contrato de operaciones que tienen que tener los controladores de ficheros en la nube.

- *GoogleDriveController*: Controlador específico de las operaciones sobre *Google Drive*.

4.3.5 Subsistema *Singletons*

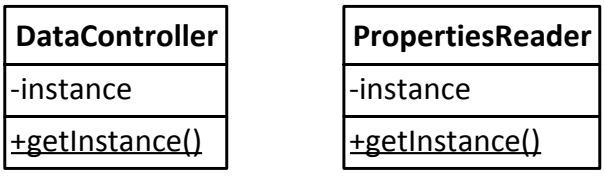


Figura V.14: Subsistema *Singletons*

El código contiene dos clases que siguen el patrón *Singleton*.

- *DataController*: Controlador del acceso a base de datos. Siguiente una estructura *multi-tier,* este controlador nos permite acceder a base de datos de forma aislada desde cualquier parte del código. Si en algún momento se cambia el sistema de persistencia el servicio no se vería afectado.

- *PropertiesReader*: Con esta clase se accede a las diferentes propiedades de configuración almacenadas en un fichero con la extensión *.properties*.

4.3.6 Subsistema *Cryptography*

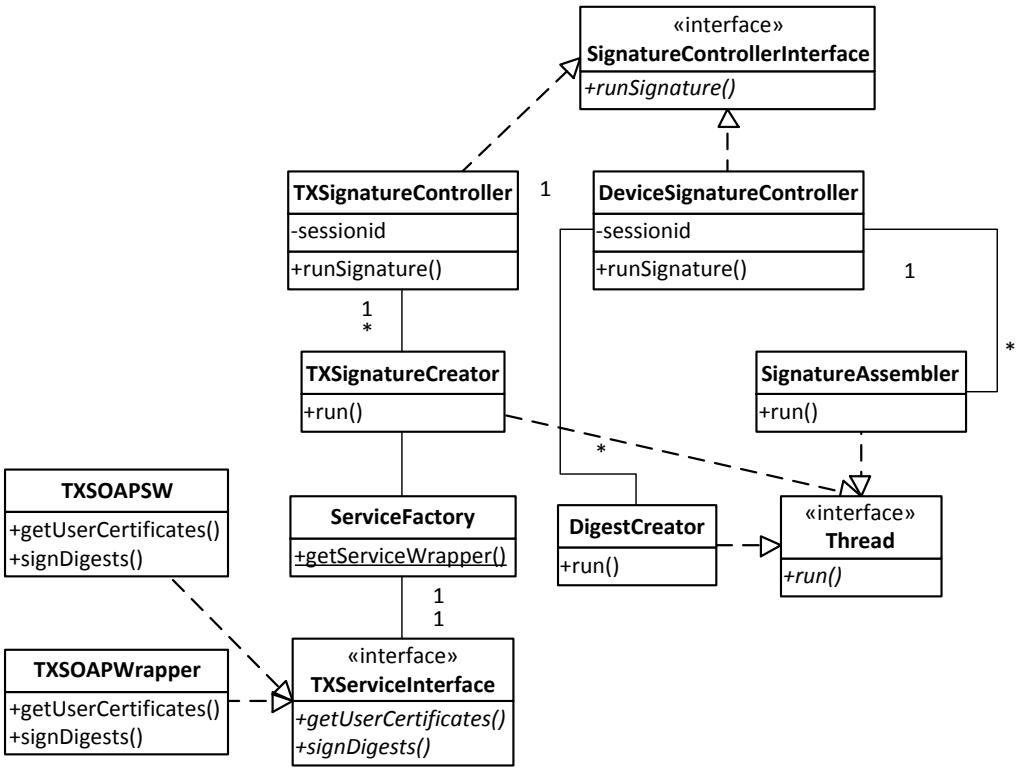


Figura V.15: Subsistema *Cryptography*

Sin duda, la parte más importante es la encargada de realizar las operaciones criptográficas. La clase principal de este subsistema, la que conecta con la clase *RestService*, es *SignatureControllerInterface*. A partir de ese punto, se deriva la estructura de clases que hacen posible la creación de firmas locales y firmas remotas.

- *SignatureControllerInterface*: Interfaz que define las operaciones que tienen que implementar los controladores de firma.

- *TXSignatureController*: Controlador de las operaciones de firma en la nube mediante el servicio externo *TrustedX*.

- *DeviceSignatureController*: Controlador de las operaciones de firma en el dispositivo móvil.

- *TXSignatureCreator*: Clase encargada de gestionar la firma de *TrustedX* y componer el documento PDF con los datos de firma recibidos. Implementa la clase *Thread* para realizar las operaciones en otro hilo de ejecución.

- *ServiceFactory*: Siguiendo un modelo *Factory Pattern*, esta clase se encarga de servir las diferentes instancias declases que realizan peticiones a *TrustedX*.

- *TXServiceInterface*: Interfaz que define las operaciones que tienen que implementar las clases que hagan peticiones a *TrustedX*. Concretamente tienen que ser capaces de obtener los certificados asociados al usuario y firmar datos.

- *TXSOAPWrapper*: Implementación de *TXServiceInterface* que realiza las peticiones a *TrustedX* mediante una encapsulación en llamadas de tipo SOAP.

- *TXSOAPSW*: Esta implementación también usa SOAP pero lo encapsula en un nivel superior utilizando la API de Safelayer *SmartWrapper*.

- *DigestCreator*: Cuando se realiza la firma en el dispositivo, esta clase se encarga del paso previo: calcular el *digest* de los datos a firmar. Implementa la clase *Thread* para realizar las operaciones en otro hilo de ejecución.

- *SignatureAssembler*: Una vez se reciben los datos de firma del dispositivo móvil, se llama a los métodos de esta clase para completar la el documento PDF firmado. Implementa la clase *Thread* para realizar las operaciones en otro hilo de ejecución.

4.4 Implementación del flujo de firma

A pesar de que el servidor no crea la firma digital en sí, tiene que preparar el documento PDF para que pueda ser firmado externamente y después componer el documento con la firma. Para ello se utiliza el *framework* de manipulación de documentos PDF iText.

El flujo divide el proceso de firma en dos pasos. El primer paso se basa en crear un nuevo documento PDF, que es una copia del original con espacio extra para añadir la firma más otros datos relacionados, calcular el *digest* de ese documento y enviarlo al servicio que se encargue de firmarlo.

El segundo paso consiste en, una vez recibida la firma, adjuntarla al documento en la posición que se le había reservado y cerrar el flujo.

Con esto se crea el documento firmado. De la validación de la firma se encarga el propio software de lectura de PDFs de Adobe.

A continuación se muestran los diagramas de secuencia de los procesos de firma.

4.4.1 Firma con el dispositivo móvil

Para realizar la firma en el dispositivo móvil el servidor primero calcula los valores de los *digest* de los documentos a firmar tal y como se muestra en el siguiente diagrama de secuencia.

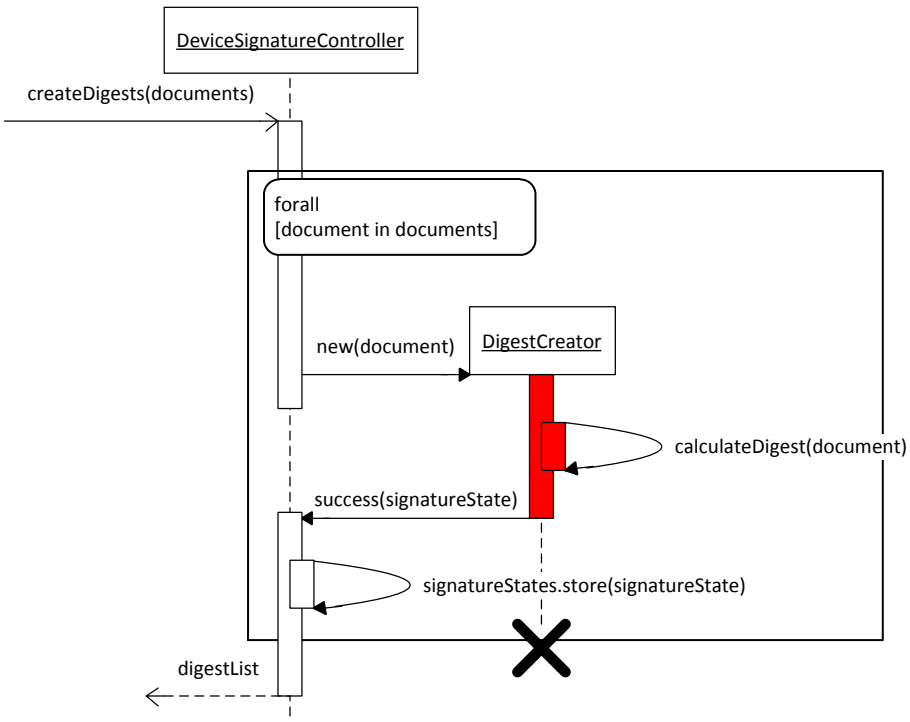


Figura V.16: Firma diferida en dispositivo: calcular *digests*

Para cada documento crea un nuevo *thread* (en rojo) que calcula el digest del documento. Si los documentos están almacenados en la nube, se descargan.

Una vez obtenido el valor, el *thread* llama a un *callback* del controlador pasándole el estado del documento creado y preparado para añadir la firma (tal y como se detalla en el apartado 1.2.1.4 "Firma PDF" del capítulo II – CONCEPTOS BÁSICOS. El controlador almacena el estado para su posterior finalización una vez obtenido el valor de la firma y devuelve la lista de todos los *digest* cuando están acabados.

El siguiente diagrama muestra cómo se finalizan las firmas una vez obtenido el valor desde el dispositivo móvil.

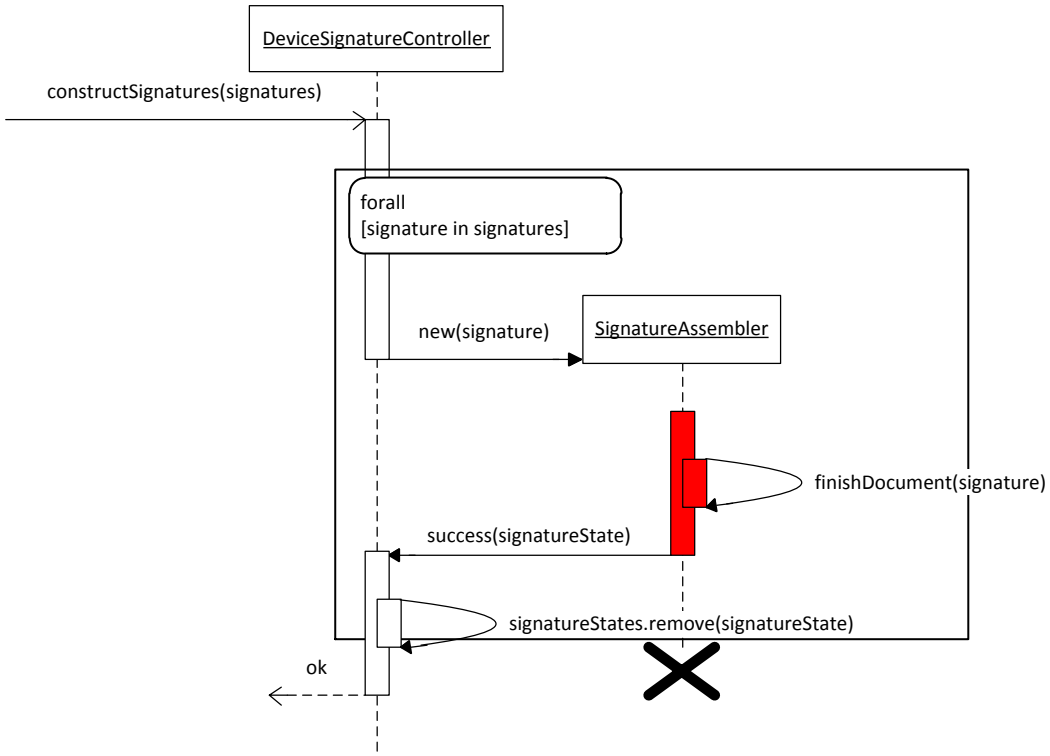


Figura V.17: Firma diferida en dispositivo: completar firma

Para cada firma recibida, el controlador crea un nuevo *thread* que añade el valor de la firma al documento creado en el paso anterior. Cuando acaba, llama a un *callback* del controlador para notificarlo y que éste elimine el estado de la firma que había almacenado. Cuando todos los estados han sido eliminados finaliza el proceso.

4.4.2 Firma CMS con *TrustedX*

Cuando la firma se realiza con *TrustedX* el flujo es ligeramente diferente al caso en el que la firma se realiza en el dispositivo móvil. Esto es porqué es el mismo servidor quien se encarga de hacer la llamada a *TrustedX* para cada documento, y una vez obtenida la firma ya se puede crear el documento firmado. El siguiente diagrama de secuencia muestra este flujo con más detalle.

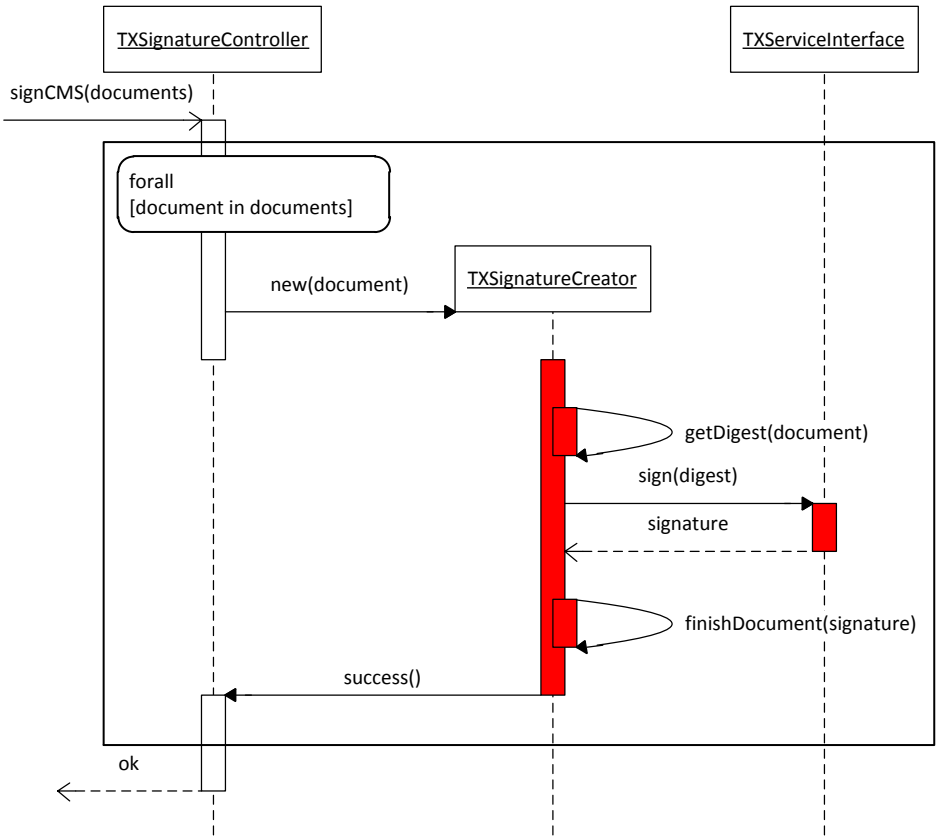


Figura V.18: Firma CMS con TrustedX

5 Desarrollo del *plugin* para *Smartphones*

En este apartado se describirá el proceso de creación del *plugin* de firma del *framework* PhoneGap para iOS y Android.

5.1 Especificación del plugin de firma

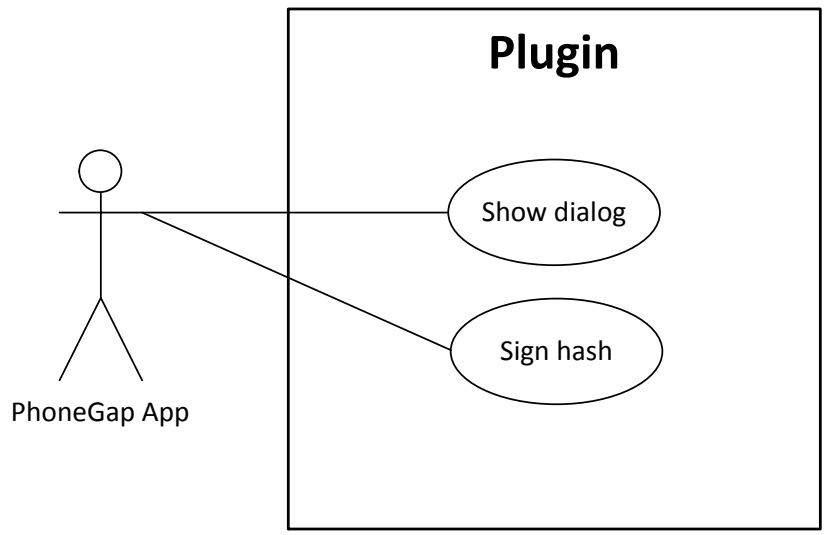
El *plugin* de firma móvil se ha diseñado para que tenga dos funcionalidades nativas. La principal es la de **firmar** unos datos de entrada codificados en base 64. La segunda, y consecuencia de la primera, es la de **obtener los certificados** almacenados en el dispositivo móvil y que contengan claves privadas asociadas para poder realizar la firma.

Dado que en el demostrador implementado se ofrece la posibilidad de firmar datos con *TrustedX* existen también las funcionalidades de obtener los certificados de *TrustedX* y firmar los datos con las claves privadas asociadas a esos certificados. Aunque como se hace vía servicio, no se requieren funcionalidades nativas para esos casos.

El *plugin* mostrará los certificados en un selector de la interfaz nativa que contendrá ambos certificados: los de *TrustedX* y los del dispositivo. Esta funcionalidad alberga algunas diferencias de una plataforma a otra debido a restricciones de implementación que se comentarán en secciones posteriores.

5.1.1 Casos de uso

Se ha hecho una especificación genérica del *plugin* desde el punto de vista de la App PhoneGap como actor principal que interactúa con el *plugin* nativo. Ésta especificación contiene dos casos de uso: *show dialog*, que muestra un diálogo/selector con los certificados para firmar y *sign hash* que devuelve el resultado de la firma de unos datos de entrada.



Al ser una especificación genérica desde el punto de vista de la App multiplataforma, el resultado varía ligeramente entre las diferentes implementaciones nativas del *plugin*, manteniéndose sin embargo, sus funcionalidades principales.

5.1.1.1 Descripción de los casos de uso

A continuación definen formalmente los casos de uso del *plugin*.

Caso de uso Show dialog

Actor	App
Precondición	Tener certificados en el dispositivo y recibir lista de certificados externos.
Postcondición	Mostrar un selector con los certificados del dispositivo y los certificados externos.
Flujo	1. El caso de uso empieza cuando la App ejecuta la llamada al *plugin* nativo pasando como parámetro una lista de nombres de certificados recibidos de un servicio externo. 2. El *plugin* crea un selector que muestra en la interfaz de la App que contiene los certificados externos y nativos. 3. El caso de uso finaliza satisfactoriamente.

Caso de uso Sign hash

Actor	App
Precondición	Tener un certificado seleccionado con clave privada asociada y unos datos de entrada codificados en base 64.
Postcondición	El resultado de firmar el hash con la clave privada del certificado seleccionado codificado en base 64.
Flujo	1. El caso de uso empieza cuando la App ejecuta la llamada al *plugin* nativo para firmar unos datos. 2. El *plugin* decodifica los datos de entrada y los firma con la clave privada del certificado seleccionado. 3. El *plugin* codifica en base 64 los bytes de firma y los devuelve a la App. 4. El caso de uso finaliza satisfactoriamente.

5.2 *Definición de la API Javascript del plugin*

Una vez finalizada la especificación se ha procedido a la implementación de la API Javascript. Ésta API encapsula las llamadas al *plugin* en métodos que facilitan su incorporación al código HTML5 de la aplicación.

PhoneGap ofrece el método *cordova.exec* para hacer las llamadas nativas. Como parámetros hay que pasar el identificador del *plugin* que se ha especificado en el fichero de configuración, el método o *action* a llamar ya que un mismo *plugin* puede contener varios métodos y un JSON con los parámetros de entrada. El *plugin* se ejecuta asíncronamente así que los *callbacks* de retorno satisfactorio y erróneo también se pasan como parámetros.

A continuación se describen los dos métodos de la API que llaman a *cordova.exec* para ejecutar el código nativo.

Método show

Descripción	Muestra un selector nativo de certificados por pantalla.
Parámetros entrada	• **certificates**: contiene los nombres de los certificados externos junto con sus identificadores en formato JSON.
Parámetros salida	• **success**: *callback* que se ejecuta cuando el *plugin* finaliza satisfactoriamente. Como parámetros lleva un texto que indica si el certificado seleccionado es externo o nativo y otro texto que si el certificado seleccionado es externo será su identificador, y si es nativo será el certificado codificado en base 64. • **fail**: *callback* que se ejecuta en caso de error. Como parámetros lleva un texto describiendo el error.

Método deviceSign

Descripción	Firma los datos de entrada y devuelve el resultado codificado en base 64.
Parámetros entrada	• **digests**: contiene los nombres de los documentos a firmar y sus *digests* asociados
Parámetros salida	• **success:** *callback* que se ejecuta una vez las firmas se han realizado. Como parámetro lleva un JSON conteniendo los nombres de los documentos y sus firmas asociadas codificadas en base 64. • **fail:** *callback* que se ejecuta en caso de error. Como parámetros lleva un texto describiendo el error.

5.3 Desarrollo del plugin para iOS

En este apartado se describirá el proceso de diseño e implementación de la versión iOS del *plugin* de firma.

La implementación se ha realizado en el lenguaje nativo del sistema iOS, que es Objective-C.

5.3.1 Diseño de la interfaz gráfica del selector

Para diseñar el selector de certificados se ha optado por utilizar el componente *picker* de iOS. Este componente se caracteriza por imitar una rueda con varios elementos, de los cuales el que está en el centro representa el seleccionado.

Se ha modificado la vista de los elementos para que puedan contener imágenes y así distinguir qué certificados pertenecen al servicio externo y qué certificados pertenecen al dispositivo.

Los que llevan el logo de *TrustedX* representan los del servicio externo, mientras que los que llevan el logo de Apple representan los del dispositivo. La siguiente captura muestra el resultado.

Figura V.19: Interfaz gráfica del selector iOS

5.3.2 Diagrama de clases

A continuación se muestra el diagrama de clases conceptual del *plugin* para iOS.

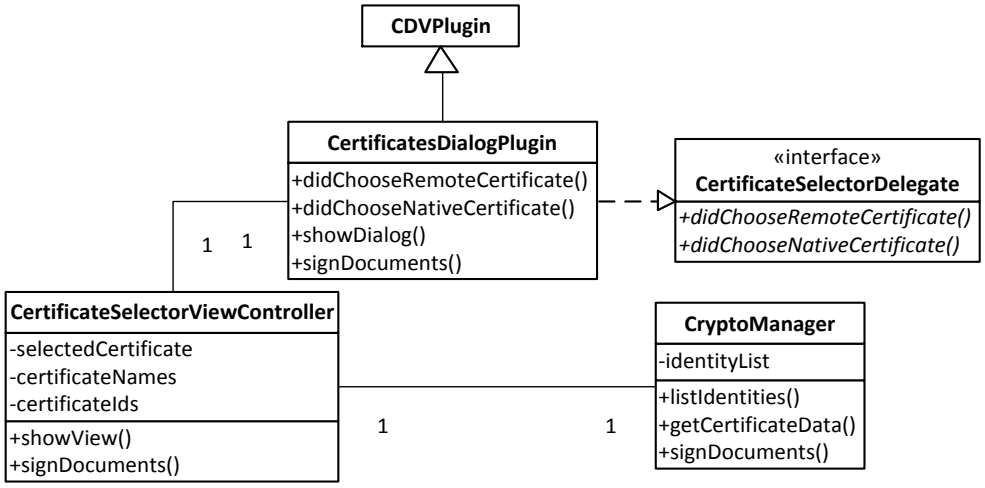


Figura V.20: Diagrama de clases del *plugin* iOS

El *plugin* se compone por tres clases principales:

- *CertificatesDialogPlugin*: clase que hereda de *CDVPlugin* perteneciente a las librerías de PhoneGap y actúa como punto de entrada al *plugin*.
- *CertificateSelectorViewController*: clase que actúa de controlador de vista para el selector de certificados.
- *CryptoManager*: clase que se encarga de los temas criptográficos, como obtener los certificados o firmar datos.

La interfaz *CertificateSelectorDelegate* es implementada por la clase que tiene que recibir los eventos relacionados con la selección de certificados.

El diseño de clases se ha realizado de forma modular de manera que se pueda incorporar en cualquier tipología de proyecto, sea híbrido o nativo.

5.3.3 Diagramas de secuencia

Los siguientes diagramas de secuencia muestran el funcionamiento de los principales procedimientos del *plugin*.

5.3.3.1 *Mostrar diálogo*

La clase principal, punto de acceso del *plugin*, crea un controlador de vista (1.) para el selector de certificados, acorde con el patrón MVC[23] que sigue Cocoa. El controlador es inicializado con una lista de certificados provenientes de un servicio externo, como puede ser *TrustedX*. Para obtener la lista de certificados nativos se crea una nueva instancia de la clase *CryptoManager* (2.) de la cual se obtiene la lista (4.) para poblar (6.) el selector con ambos tipos de certificados.

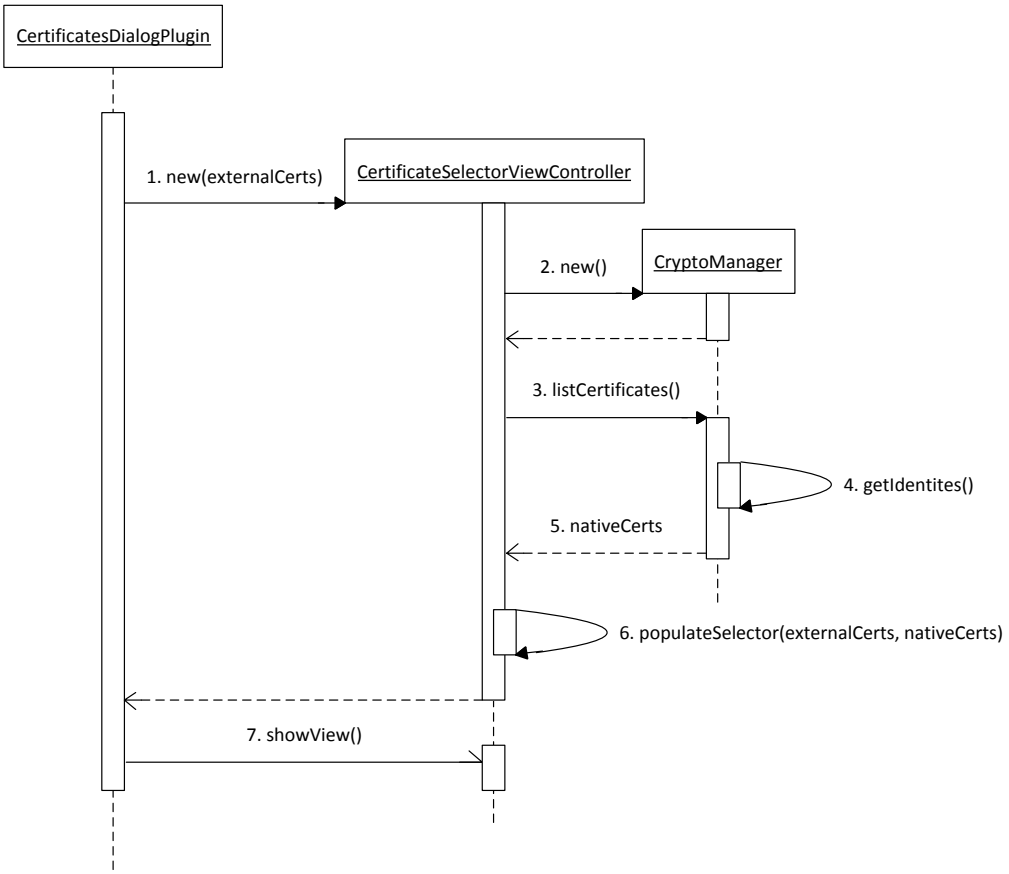


Figura V.21: Operación “Mostrar diálogo”

[23] Model View Controller

5.3.3.2 Seleccionar certificado externo

Cuando se pulsa sobre un certificado del servicio externo (1.), el controlador de vista únicamente tiene que devolver su identificador (2.) al *plugin* mediante un *callback* (3.).

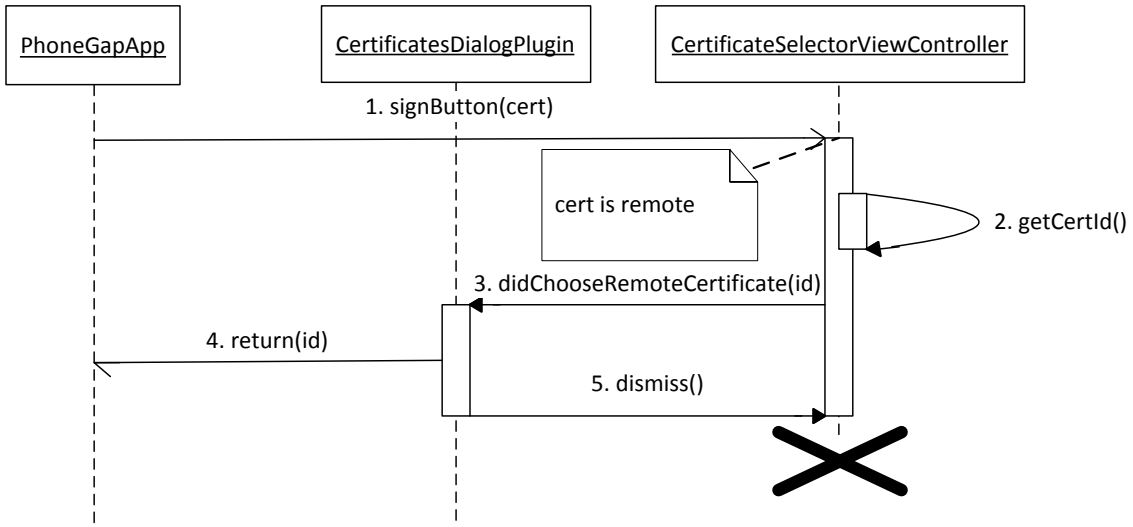


Figura V.22: Operación "Seleccionar certificado externo"

5.3.3.3 Seleccionar certificado nativo

Cuando se pulsa sobre un certificado nativo (1.), el controlador llama a la instancia de *CryptoManager* (previamente inicializada) para que obtenga los datos del certificado (3.). El controlador devuelve esos datos a *CertificatesDialogPlugin* mediante un *callback* (6.).

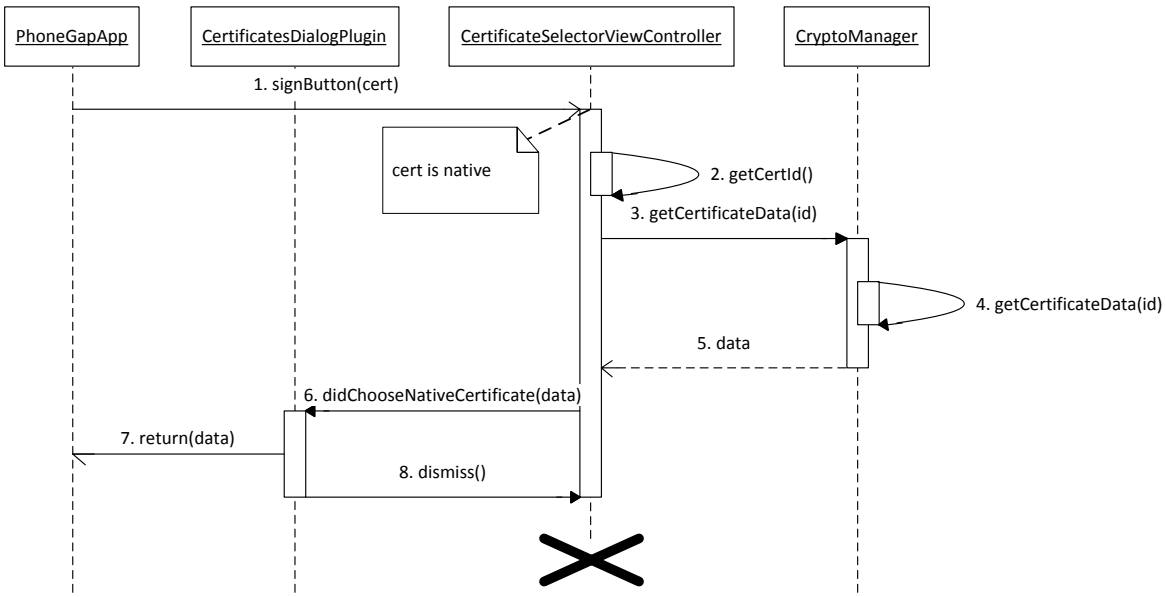


Figura V.23: Operación "Seleccionar certificado nativo"

5.3.3.4 Firmar datos con clave del dispositivo

En el flujo de firma, *CertificatesDialogPlugin* llama al método *signDocuments* (1.) del controlador de vista pasándole la lista de *digests* a firmar. Éste llama al método para firmar de la instancia previamente creada de *CryptoManager* (2.). Los parámetros de éste método son la lista de *digest* junto con el identificador de certificado que se ha seleccionado para la firma.

CryptoManager crea un thread (3.) para cada *digest* y computa (6.) su firma. Una vez todas las firmas se han completado concurrentemente, recoge los resultados (7.) y los devuelve al controlador de vista (8.), quien a su vez, lo devuelve al *plugin* (9.).

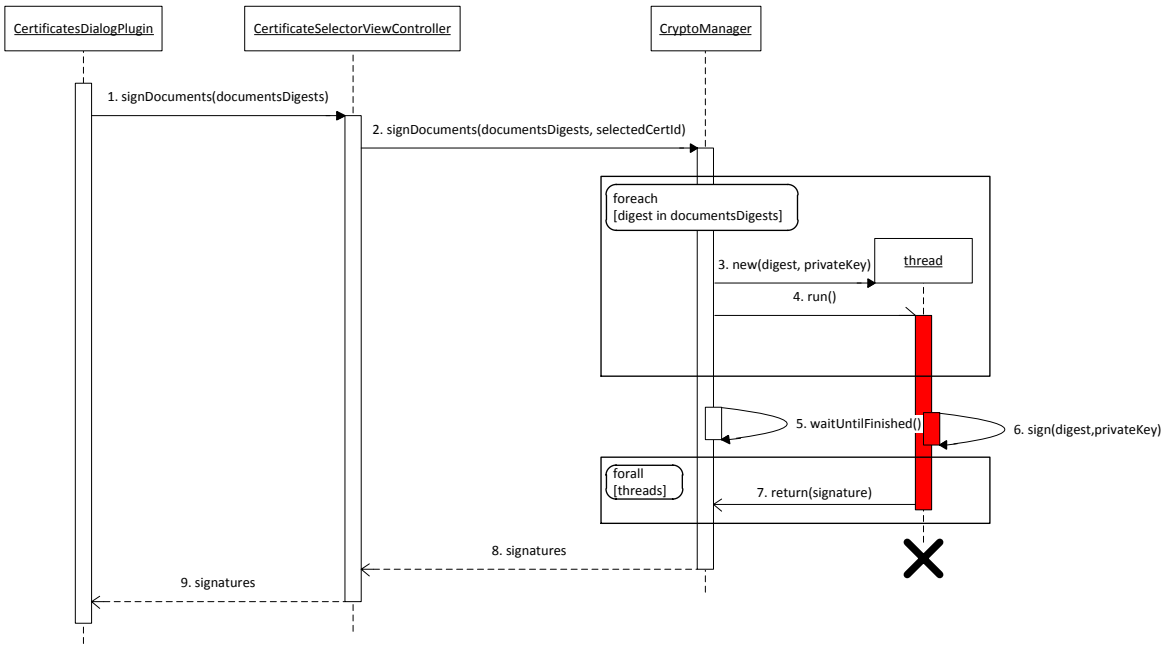


Figura V.24: Operación "Firmar datos con clave nativa"

5.4 Desarrollo del plugin para Android

En este apartado se describirá el proceso de diseño y posterior implementación del *plugin* para sistemas Android.

La implementación se ha hecho en Java, el lenguaje de desarrollo nativo para el sistema de Google.

5.4.1 Diseño de la interfaz gráfica del selector

Las versiones de Android 4.0 en adelante son las que permiten acceder a los certificados de sistema desde **cualquier aplicación**. Esto se hace mediante una llamada nativa que muestra un diálogo de sistema con todos los certificados, como el de la siguiente captura.

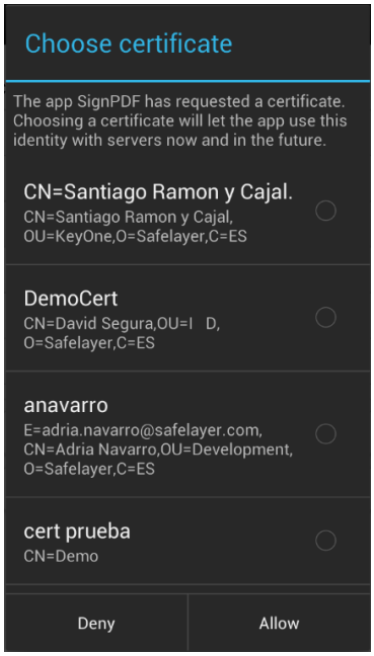


Figura V.25: Selector nativo de Android

Esto ha condicionado el diseño de la interfaz, dado que no es posible obtener una lista de los certificados para crear un selector personalizado. Para obtener un certificado hay que pasar siempre por el selector nativo.

La solución propuesta ha intentado que el resultado sea lo menos intrusivo posible para el usuario. Consiste en crear un selector para los certificados externos lo más parecido posible al de certificados de sistema, e incorporar un botón para abrir el selector nativo, si el usuario prefiere escoger un certificado del dispositivo.

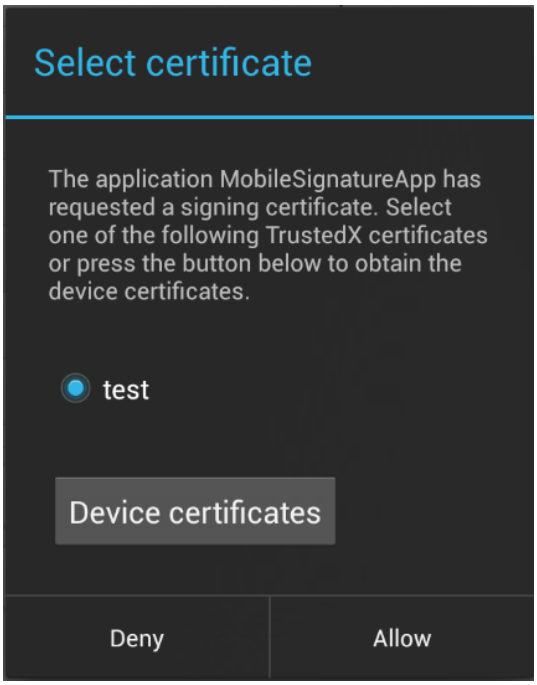


Figura V.26: Selector de certificados externos

La captura muestra el diálogo resultante con un texto y apariencia parecidos al diálogo del selector nativo. Al pulsar el botón *Device certificates* se mostraría el diálogo nativo anterior tal y como se detalla en el siguiente diagrama de flujo.

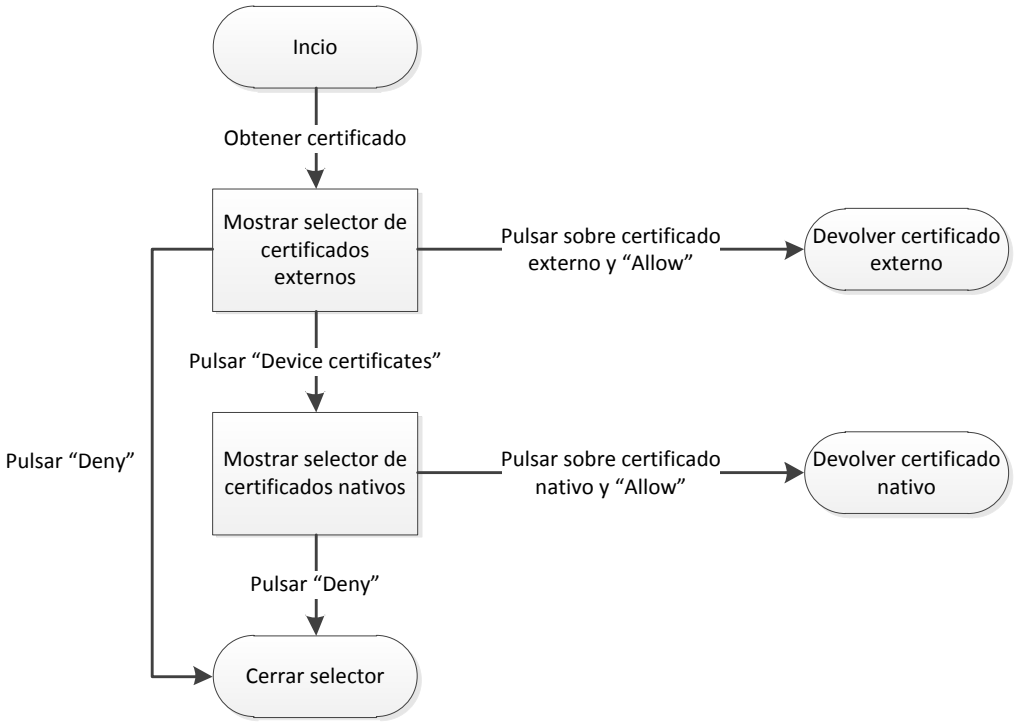


Figura V.27: Flujo del selector del *plugin*

5.4.2 Diagrama de clases

El siguiente diagrama de clases muestra la arquitectura del *plugin*, muy similar a la del sistema iOS.

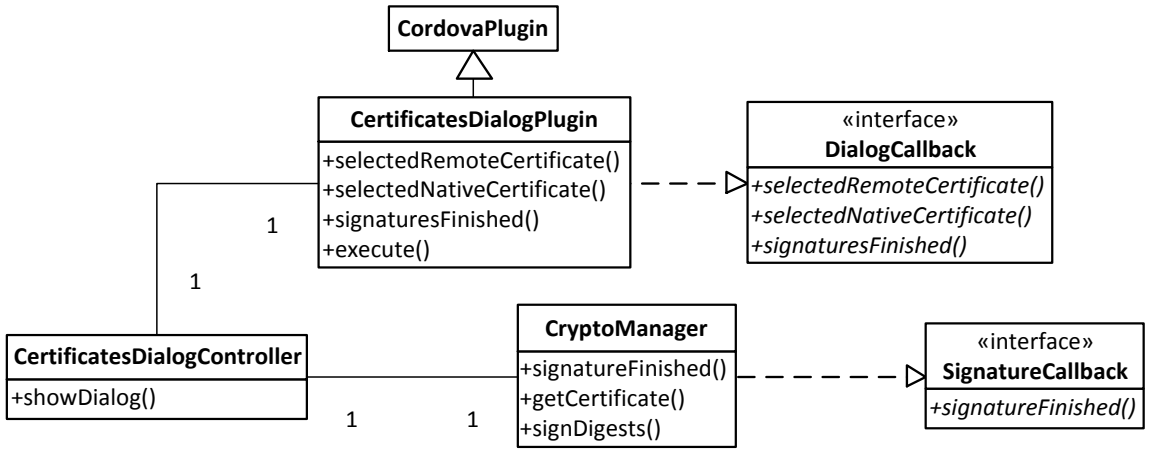


Figura V.28: Diagrama de clases del *plugin* Android

Se compone de 3 clases principales:

- *CertificatesDialogPlugin*: Clase que hereda de Cordova*Plugin*, perteneciente a las librerías de PhoneGap, y actúa de punto de entrada al *plugin*.
- *CertificatesDialogController*: Controlador para la interfaz del selector.
- *CryptoManager*: Controlador de la parte criptográfica.

Los interfaces *DialogCallback* y *SignatureCallback* son implementados por las clases que tienen que recibir la respuesta de las interacciones del usuario y las operaciones de firma, respectivamente.

5.4.3 Diagramas de secuencia

En esta sección se muestran los diagramas de secuencia de las operaciones más relevantes de la versión Android del *plugin* de firma.

5.4.3.1 Mostrar diálogo

Para mostrar el diálogo con el selector de certificados, el *plugin* recibe (1.) una lista de nombres de certificados externos, que pasa al controlador de vista (2.), que crea el diálogo (3., 4.). Los certificados nativos no se muestran debido a la limitación de Android de tener que pasar por el diálogo del sistema.

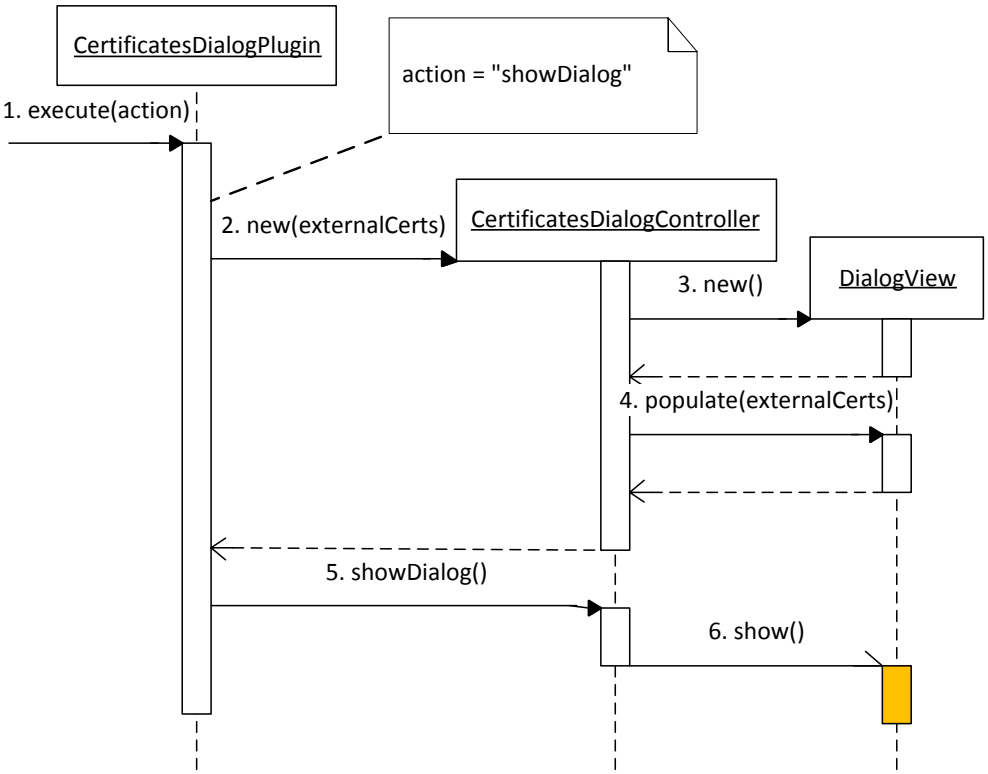


Figura V.29: Operación "Mostrar diálogo"

5.4.3.2 Seleccionar certificado externo

Cuando el usuario, a través de la App selecciona un certificado (1.) perteneciente al servicio externo, el controlador de vista obtiene su identificador asociado (3.), cierra el diálogo (4.) y a través de un *callback* (5.) lo devuelve a la clase principal del *plugin*.

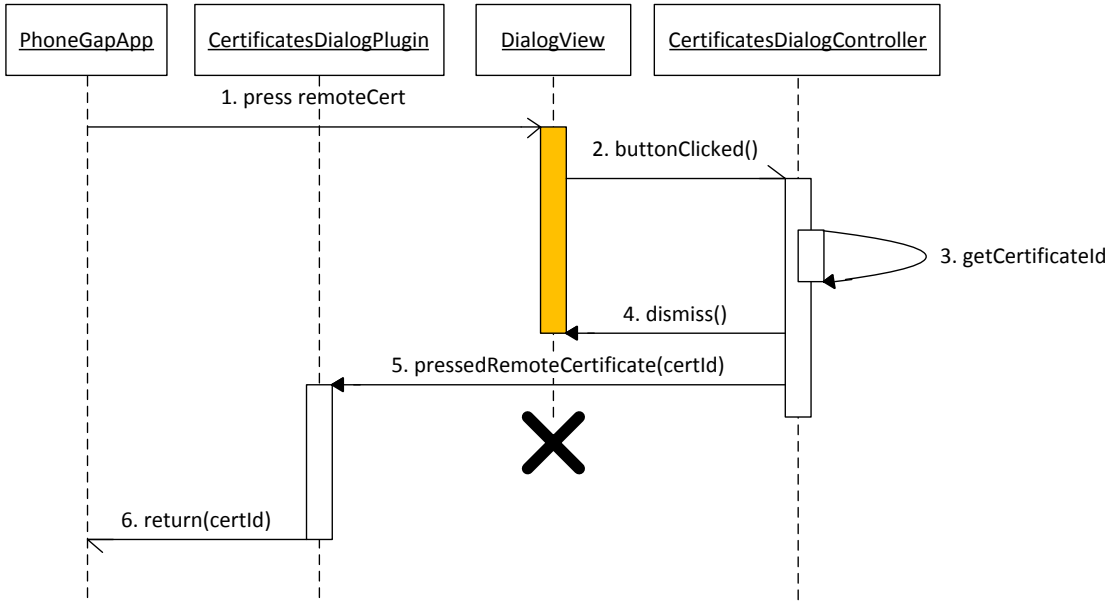


Figura V.30: Operación "Seleccionar certificado externo"

5.4.3.3 Seleccionar certificado nativo

Cuando el usuario pulsa el botón del diálogo del *plugin* para obtener los certificados del dispositivo (1.), el controlador crea una nueva instancia de *CryptoManager* (4.) que a través de la clase nativa *KeyChain* muestra el selector de certificados nativos (6.). Cuando el usuario pulsa sobre uno (7.), un *callback* devuelve el alias a *CryptoManager* (8.), que obtiene sus datos en base 64 (12., 13.) y, a través de otro *callback* (14.), los pasa a la clase principal del *plugin*. *CryptoManager* se guarda una referencia a la clave privada seleccionada (11.) para poder firmar los datos una vez los reciba.

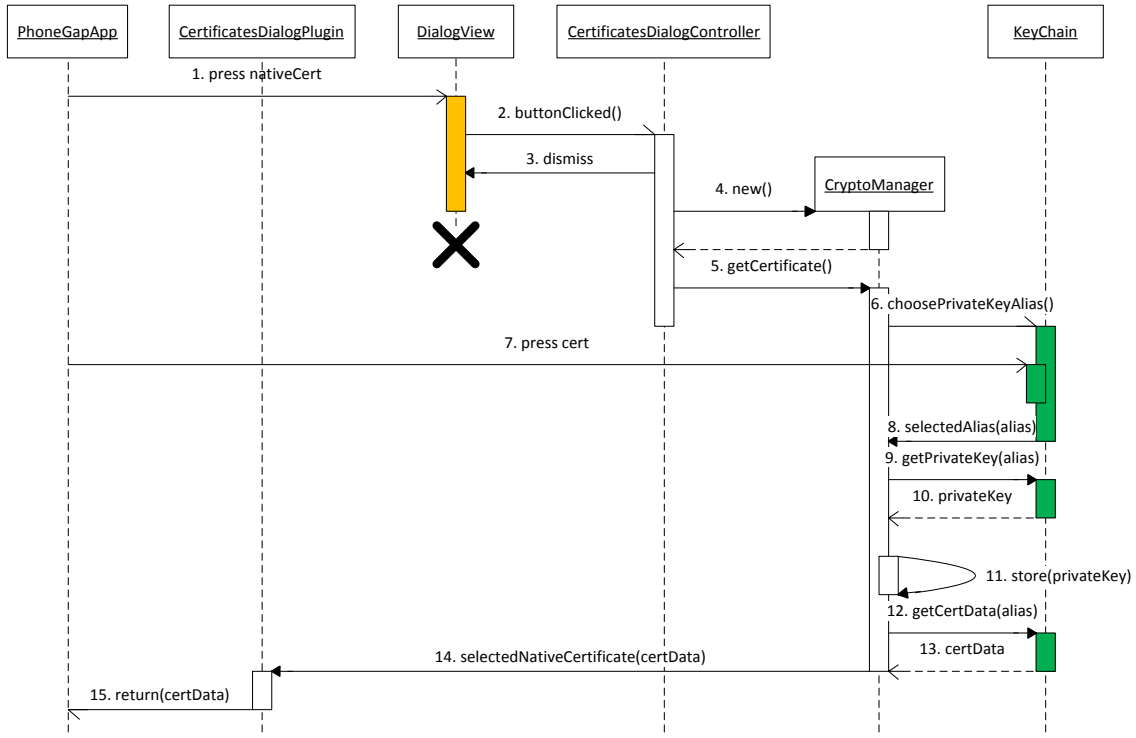


Figura V.31: Operación "Seleccionar certificado nativo"

5.4.3.4 Firmar datos con clave del dispositivo

Éste procedimiento ocurre cuando el usuario ya ha seleccionado un certificado nativo, el certificado ha sido mandado al servidor, que devuelve los *digest* de los datos a firmar y el dispositivo lo ha recibido. Es el controlador de vista quien llama a la instancia previamente creada de *CryptoManager* (3.), que se ha guardado una referencia a la clave privada de firma, pasándole una lista con los *digest*.

CryptoManager itera sobre la lista creando un nuevo *thread* para cada elemento (4., 5.). Cada *thread* firma sus datos (6.) y cuando finaliza llama a un *callback* pasándole los datos pertenecientes a la firma (7.). Un contador se encarga de controlar que todas las firmas han acabado (8.) para llamar a un *callback* con todos los resultados de las firmas en una lista (9.).

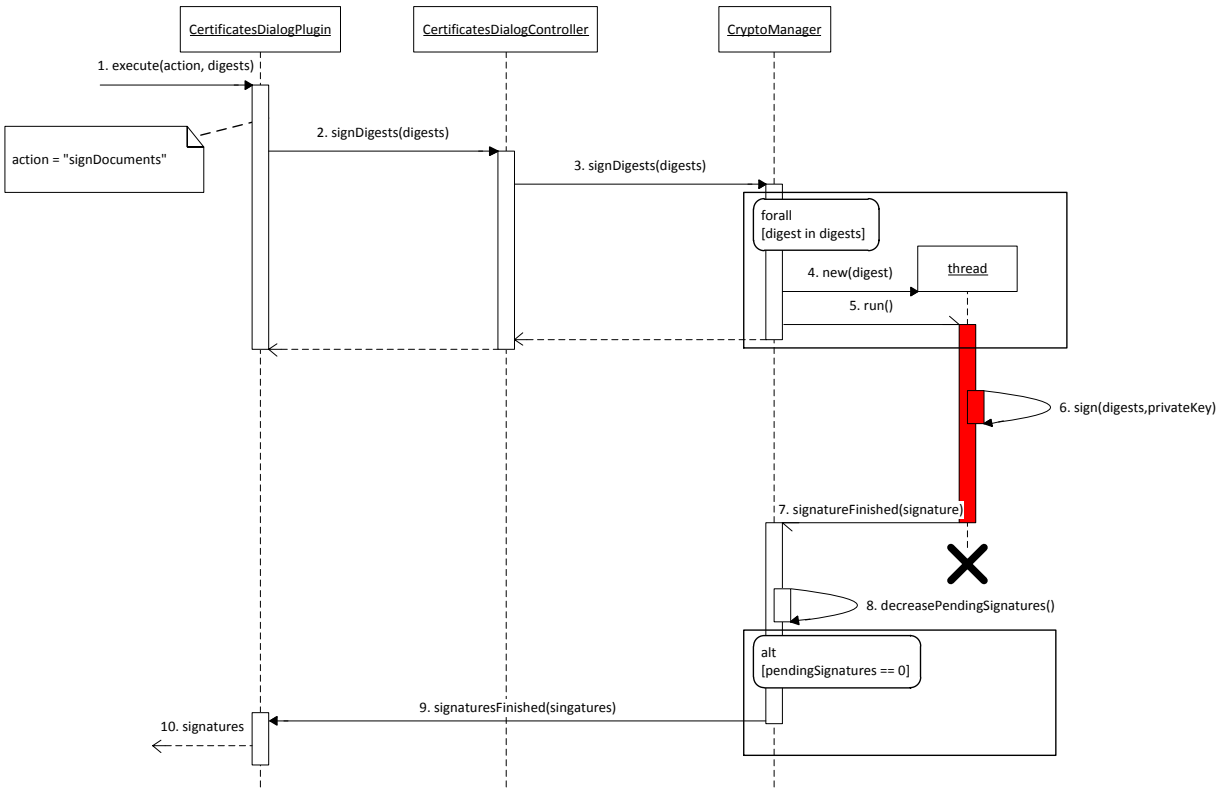


Figura V.32: Operación "Firmar datos con clave nativa"

VI - PLANIFICACIÓN

En este capítulo se detallará la planificación del proyecto. Primero se verá un diagrama de Gantt con una planificación temporal del proyecto, desde su inicio hasta su fin. Después se verá una estimación de los costes económicos del proyecto.

1 Planificación temporal

El proyecto se inició en abril de 2012 con aún aspectos por definir y se finalizaría en marzo del año siguiente con la lectura ante el tribunal. Se pensó dividir el proyecto en las siguientes etapas:

- **Investigación y formación**: Ésta etapa consiste en adquirir una sólida base sobre las tecnologías y protocolos de seguridad como PKI, OAuth, etc. así como de los productos de la empresa y su funcionamiento, como *TrustedX*. También se aprenden tecnologías requeridas para el desarrollo del proyecto como la programación en dispositivos iOS y el desarrollo mediante CPT.

- **Especificación**: Durante la etapa de especificación se define el proyecto conceptualmente y se divide en dos etapas:

    - **Diseño de la aplicación móvil**: Ésta etapa especifica cómo va a ser el *framework* y sus funcionalidades. También define la aplicación móvil que va a actuar de demostrador del *framework*.

    - **Diseño del servidor**: Para dar una especificación del servidor se definieron los casos de uso de sus funcionalidades principales.

- **Implementación**: En ésta etapa se desarrolla el proyecto y se divide otra vez en las siguientes etapas que durante algún tiempo se van desarrollando en paralelo:

    - **Implementación de la aplicación móvil**: Se crea el código de los *plugins* del *framework* PhoneGap así como de la aplicación demostradora.

    - **Implementación del servidor**: Se estudia el *framework* iText para realizar las operaciones criptográficas con los documentos PDF y se implementan sus funcionalidades.

- **Pruebas funcionales del sistema**: Una vez se finaliza la implementación de todo el sistema se ejecutan pruebas funcionales para validar su correcto funcionamiento.

- **Documentación:** La etapa de documentación trascurre en paralelo durante todo el desarrollo del proyecto, ya que se van generando documentos e informes que luego son aplicables a la memoria. Una vez se finaliza la memoria se empieza a preparar la presentación que se usará durante la defensa.

Como ya estaba previsto, las primeras etapas de formación e investigación se alargaron considerablemente ya que había que estudiar a fondo las tecnologías y protocolos de seguridad, especialmente basadas en PKI y otros relacionados con la privacidad como OAuth, que resultan de vital importancia dado el tema del proyecto y para entender las tecnologías de la empresa.

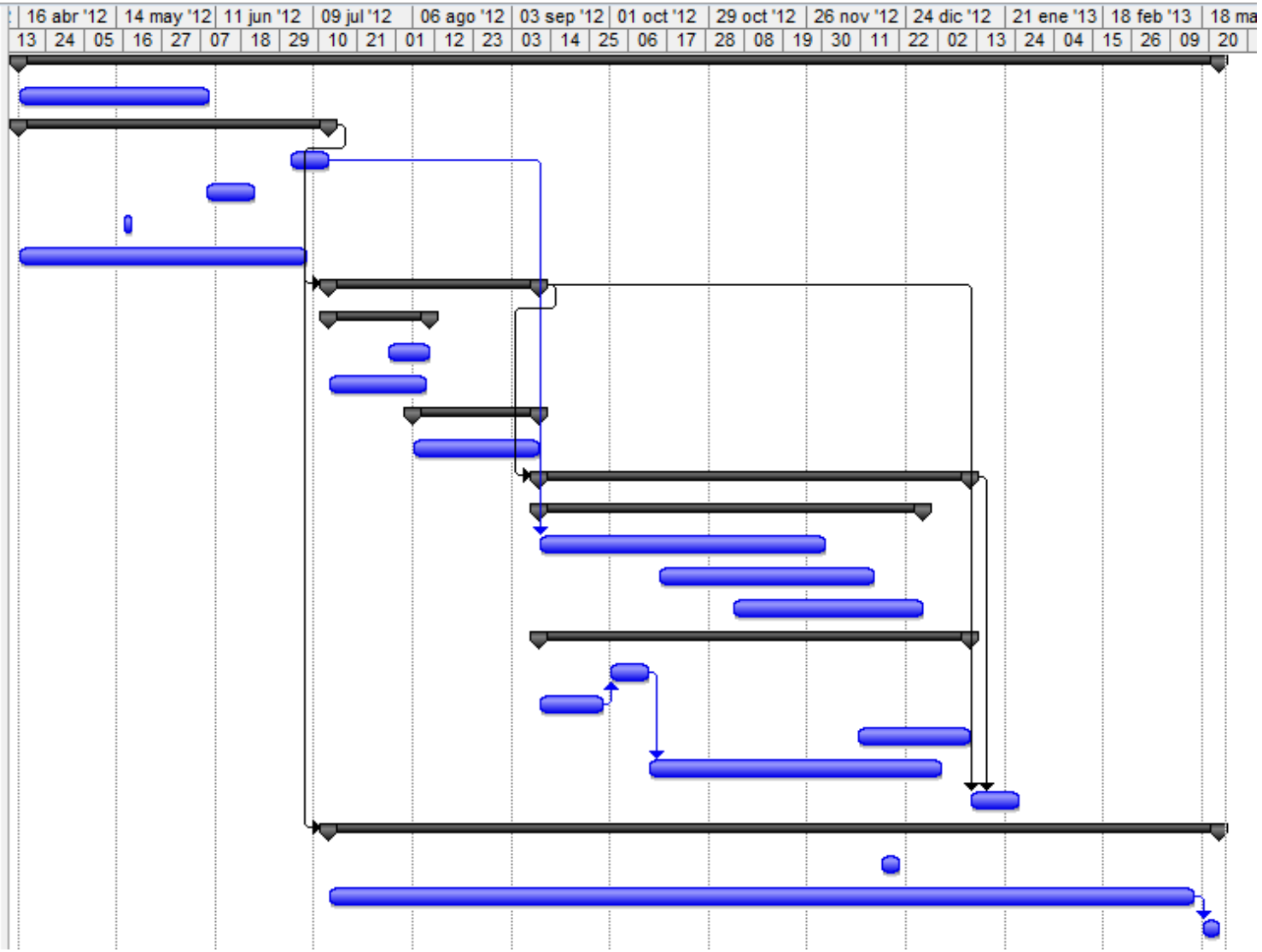
Antes de empezar siquiera la especificación se empezó a experimentar con diferentes tecnologías y la PKI para empezar a orientar la temática del proyecto. Cuando se decidió atacar la movilidad empezó el estudio de las CPT y sus capacidades frente a la seguridad y la firma electrónica.

Dado que es un proyecto de investigación han sido necesarios cerca de tres meses para adquirir la formación necesaria y realizar los experimentos y pruebas requeridas. En la planificación inicial ya estaba prevista una duración larga de las etapas de formación, investigación, y especificación (en total 255+280 h), de la cual dependerían el proyecto y por tanto el resto de etapas. La siguiente tabla muestra las horas finalmente dedicadas al proyecto por etapas.

Proyecto			
Etapa	**Inicio**	**Fin**	**Horas**
Definición del Proyecto y Formación	16/04/12	12/07/12	255
Especificación	13/07/12	10/09/12	280
Implementación	11/09/12	10/01/13	430
Pruebas funcionales	11/01/13	24/01/13	70
Documentación	13/07/12	22/03/13	200
TOTAL	**16/04/12**	**22/03/13**	**1235**

PLANIFICACIÓN

Nombre de tarea	Duración
Realización del proyecto	**245 días?**
Definición del proyecto	2 mss
Investigación y formación	**64 días?**
Asistencia a curso de iOS	9 días
Estudio de CPT	2 sem.
Curso de TrustedX	2 días
Tecnologías PKI	3 mss?
Especificación	**42 días?**
Diseño de la aplicación móvil	**21 días**
Diseño de las pantallas	2 sem.
Especificación de los plugins	1 ms
Diseño del servidor	**26 días?**
Especificación de los casos de uso	26 días?
Implementación	**88 días?**
Implementación de la aplicación móv	**79 días?**
Implementación del plugin de iOS	59 días?
Implementación del plugin de Android	45 días?
Implementación de la aplicación JavaSc	40 días?
Implementación del servidor	**88 días?**
Estudio y pruebas del framework iText	9 días?
Implementación de la autenticación Oau	14 días?
Integración con Google Drive	24 días?
Implementación de los casos de uso	59 días?
Pruebas funcionales del sistema	2 sem.
Documentación	**181 días?**
Informe previo del proyecto	5 días?
Memória del proyecto	176 días?
Preparación de la presentación	1 sem



2 Análisis económico

Para calcular el coste económico aproximado del proyecto se deben desglosar los distintos elementos que han intervenido en su desarrollo, desde los recursos humanos hasta el mobiliario utilizado.

Coste general

Concepto	Coste (€)
Recursos humanos	11.857'15
Equipos informáticos	1377'43
Sistemas y programas	0
Mobiliario	300
TOTAL	**13534'58 €**

Los cálculos de esta tabla se desglosan de la siguiente manera:

Recursos humanos

En el proyecto ha participado un becario en régimen de convenio universidad empresa (beca UPC), y no se tiene en cuenta el soporte y dedicación de otras personas, en este caso de la empresa. Para calcular su salario es necesario multiplicar el número de horas dedicadas por el salario base:

1 becario * 1235 horas * 8€/h = 9.880 €

Además, a este coste se deben añadir el coste adicional que supone la manutención de un empleado económicamente (luz, teléfono, etc.) que está fijada en un 20% del coste del empleado.

9.880 € + 20% = 11.856 €

11.856 € + 1'147 € gestión del convenio = 11.857'15 €

TOTAL	**11.857'15 €**

Equipos informáticos

Para el desarrollo del proyecto se han utilizado los siguientes dispositivos y equipos:

- Equipo Dell Precision T3500
- Macbook Pro
- HTC Magic
- iPhone 4
- Galaxy Nexus

Dado que todos los ítems son nuevos y la amortización que se hace en la empresa es de 3 años, para obtener su valor final amortizado hay que multiplicar el coste original por los 11 meses que se ha utilizado y dividirlo por 36.

Ítem	Coste	Valor Amortizado
Dell Precision	1.800 €	550 €
Macbook Pro	1.229 €	375'53 €
HTC Magic	199 €	60'80 €
iPhone 4	600 €	183'33 €
Galaxy Nexus	680 €	207'77 €

TOTAL	**1377'43 €**

Sistemas y programas

Para el desarrollo de la aplicación no ha sido necesario adquirir ninguna licencia puesto que las soluciones utilizadas eran en su totalidad software de libre distribución o software propietario de la empresa.

En cuanto a sistemas operativos y suites de ofimática se han usado Microsoft Windows 7 y Microsoft Office 2010, respectivamente. Ambos con licencia OEM, por lo que su precio venía incluido en el de la estación de trabajo y consideraremos un coste de 0 €.

TOTAL	**0 €**

Mobiliario

Finalmente se tiene en cuenta el mobiliario que ha sido necesario durante el desarrollo del proyecto. El material de oficina queda definido en el apartado 4 de los elementos comunes del impuesto de sociedades con un coeficiente lineal máximo del 10% durante un periodo máximo de 10 años.

Se considera:

Escritorio y silla de oficina valorados en 500 € de 4 años de antigüedad.

*Amortización: 500 * 0.1 * 4 = 200 €*

Coste: 500 - 200 = 300 €

TOTAL	**300 €**

VII - CONCLUSIONES

El objetivo de este proyecto era la creación de un *framework* de desarrollo con funcionalidades de privacidad y seguridad para móviles. Concretamente se ha extendido un *framework* ya existente llamado PhoneGap, con funcionalidades de firma digital.

Para poder realizar la extensión de este *framework* primero ha sido necesario valorar si las herramientas de desarrollo multiplataforma o CPT (*Cross-Platform Tools*) eran viables en el mercado del desarrollo para dispositivos móviles, y segundo, se ha requerido estudiar y analizar las CPT más populares.

Para realizar el análisis de las CPT primero se ha probado una muestra de aplicaciones desarrolladas con cada uno, analizando su rendimiento y posibilidades. Después se han visto como éran sus herramientas de desarrollo y que funcionalidades ofrecían.

El CPT elegido ha sido PhoneGap por su accesibilidad de precio y sus posibilidades de extensión mediante el desarrollo de *plugins*. A partir de la elección del *framework* ha empezado el desarrollo del *plugin* de firma para iOS y Android así como de la aplicación móvil demostradora y del servidor intermedio.

El resultado final ha sido una aplicación móvil que demuestra las capacidades del *framework* de firma digital integrado en el *plugin* de PhoneGap, que realiza firmas digitales tanto en el dispositivo como en el servicio *TrustedX*. El servidor es el encargado de gestionar los documentos almacenados en *Google Drive* así como de, mediante los datos de firma, componer los documentos PDF firmados.

Realizar este trabajo ha supuesto una adquisición de nuevos conceptos tecnológicos y metodologías. Ha sido necesario un ámplio estudio de los conceptos de la PKI, OAuth y de las tecnologías de Safelayer. También se han adquirido conocimientos y experiencia en los lenguajes de programación Objective-C, Java y de las tecnologías web JavaScript, HTML y CSS.

El desarrollo de un proyecto en un ámbito empresarial ha supuesto una experiencia muy enriquecedora ya que, a diferencia de los proyectos académicos, en este a priori no había nada definido, y han sido requeridos todos los estudios comentados en párrafos anteriores para dibujar la línea que iba a seguir el proyecto.

Poder usar parte del producto *TrustedX*, una de las tecnologías de seguridad más punteras del mercado desarrollado por la própia empresa a supuesto una

oportunidad para trabajar con una solución real y de ver y entender su funcionamiento, además de ser explicado por sus própios desarrolladores.

1 Líneas futuras de desarrollo

Una de las ideas iniciales que no se ha llegado a implementar era la posibilidad de incluir otro servicio de almacenamiento en la nube aparte de *Google Drive*, en concreto se pensó en *Dropbox*. Sin embargo dado que esta es una funcionalidad de una aplicación que únicamente sirve para demostrar las funcionalidades del *framework* desarrollado se le restó prioridad para en un final no llegarse a implementar.

Otra línea futura de desarrollo más útil es la de rehacer el *framework* de firma para añadirle otras funcionalidades de PKI e integrarlo con más servicios de la empresa. Esto implicaría hacer un desarrollo ya orientado a clientes reales, en lugar de a un contexto académico. También se consideraría la eliminación del requisito del servidor intermedio, ya que se trata de un servicio completamente dedicado y a medida y eso dificultaría la integración del producto.

VIII - REFERENCIAS

En este capítulo se listan las referencias del proyecto.

Estudio de las *Cross-Platform Tools* y dispositivos móviles

- *Estudio CPT,* <http://www.visionmobile.com/rsc/researchreports/VisionMobile_Cross-Platform_Developer_Tools_2012.pdf>
- *PhoneGap,*<http://phonegap.com/>
- *Xamarin*, <http://xamarin.com/>
- *Sencha*, <http://www.sencha.com/>
- *"Thoughts on Flash" Steve Jobs,*<http://www.apple.com/hotnews/thoughts-on-flash/>
- *Estadísticas aplicaciones Android,* <http://www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices>
- *Estadísticas dispositivos Android*, <http://news.cnet.com/8301-1035_3-57510994-94/google-500-million-android-devices-activated/>
- *Estadísticas AppStore*, <http://148apps.biz/app-store-metrics/>
- *Estadísticas dispositivos iOS*, <http://techcrunch.com/2012/09/12/apple-has-sold-over-400-million-ios-devices-9-5-growth-since-march/>

Criptografía y autenticación

- *iText*, <http://itextpdf.com/>
- *Android crypto*, <http://android-developers.blogspot.com.es/2012/03/unifying-key-store-access-in-ics.html>
- *OAuth 2.0,*<http://tools.ietf.org/html/rfc6749>
- *Estándares criprográficos,* <http://searchsecurity.techtarget.com/definition/Public-Key-Cryptography-Standards>
- *SAML y OAuth 2.0*, <http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-15>

- *Firmas digitales en documentos PDF*, <http://www.adobe.com/it/security/pdfs/DigitalSignaturesInPDF.pdf>

Tecnologías de desarrollo

- *JQuery Mobile*, <http://jquerymobile.com/>
- *Google Drive SDK*, <https://developers.google.com/drive/>
- *Librerías iOS*, <http://developer.apple.com/library/ios>
- *Servidor Tomcat*, <http://tomcat.apache.org/>
- JavaScript: The Definitive Guide 5th Edition, David Flanagan
- Programación iOS5, Ron Napier y Mugunth Kumar